

INFORMATION TO USERS

While the most advanced technology has been used to photograph and reproduce this manuscript, the quality of the reproduction is heavily dependent upon the quality of the material submitted. For example:

- Manuscript pages may have indistinct print. In such cases, the best available copy has been filmed.
- Manuscripts may not always be complete. In such cases, a note will indicate that it is not possible to obtain missing pages.
- Copyrighted material may have been removed from the manuscript. In such cases, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, and charts) are photographed by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each oversize page is also filmed as one exposure and is available, for an additional charge, as a standard 35mm slide or as a 17"x 23" black and white photographic print.

Most photographs reproduce acceptably on positive microfilm or microfiche but lack the clarity on xerographic copies made from the microfilm. For an additional charge, 35mm slides of 6"x 9" black and white photographic prints are available for any photographs or illustrations that cannot be reproduced satisfactorily by xerography.

Order Number 8724574

Concept formation by heuristic classification

Hadžikadić, Mirsad, Ph.D.

Southern Methodist University, 1987

Copyright ©1987 by Hadžikadić, Mirsad. All rights reserved.

U·M·I

300 N. Zeeb Rd.
Ann Arbor, MI 48106

CONCEPT FORMATION BY HEURISTIC CLASSIFICATION

A Dissertation Presented to the Graduate Faculty of

School of Engineering and Applied Science

of

Southern Methodist University

in

Partial Fulfillment of the Requirements

for the degree of

Doctor of Philosophy

with a

Major in Computer Science

by

Mirsad Hadžikadić

(B.S., University of Banja Luka, 1977)

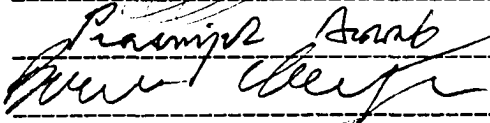
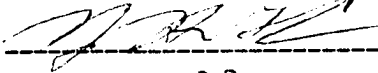
(M.S., University of Banja Luka, 1981)

July 7, 1987

✓

CONCEPT FORMATION BY HEURISTIC CLASSIFICATION

Approved by:


Pramjeet Singh

Anurag

Craig W Thompson

COPYRIGHT 1987

Mirsad Hadžikadić

All Rights Reserved

Hadžikadić, Mirsad

B.S., University of Banja Luka, 1977

M.S., University of Banja Luka, 1981

CONCEPT FORMATION BY HEURISTIC CLASSIFICATION

Advisor: Professor David Y. Y. Yun

Doctor of Philosophy degree conferred August 8, 1987

Dissertation completed July 7, 1987

This study describes an approach to the classification of a set of given events so that the resulting classification is useful for the solution of a problem at hand. A model of cognitive systems is described from a perspective of the classification process. It is used to identify sources of additional information to guide the process of classification. A classification algorithm is then devised and implemented. The algorithm is tested and evaluated in the domain of the UNIX user commands.

The algorithm reflects the heuristic nature of the classification process. Both the clustering-evaluation criterion and the update mechanism are defined in a heuristic fashion. The implemented update mechanism represents an effective learning component of the algorithm. In addition, the method for incremental update of the concept description is described. Because of the implemented approach to classification and the modified knowledge representation formalism in this work, the seriousness of the noise handling problem is significantly reduced.

TABLE OF CONTENTS

ABSTRACT	v
LIST OF ILLUSTRATIONS	x
LIST OF TABLES	xi
ACKNOWLEDGMENTS	xii
 CHAPTER 1: INTRODUCTION	 1
1.1. Characterization of Classification	1
1.2. Machine Learning and Classification	2
1.2.1. Rote Learning	3
1.2.2. Learning by Instruction	3
1.2.3. Deductive Learning	4
1.2.4. Learning by Analogy	4
1.2.5. Inductive Learning	4
1.2.5.1. Learning from Examples	5
1.2.5.2. Learning by Observation and Discovery	6
1.2.6. The Components of Classification	6
1.2.7. Related Work in Machine Learning	9
1.3. Related Work in Psychology	13
1.4. Approach Taken in this Work	16
1.5. Results of the Dissertation	18
1.6. An Example	18
1.7. Outline of the Dissertation	20
 CHAPTER 2: RESEARCH FOUNDATIONS	 21
2.1. A Goal of the Research	21
2.2. Classification as a Model-Based Process	22
2.3. Classification as a Goal-Driven Process	24
2.4. Classification Combining Model- and Data-Driven Processes	26
2.5. Background Knowledge	28
2.5.1. Goal-Dependency Network	28
2.5.2. Associative Links	31
2.5.3. The Structure of Attribute Domains	32
2.5.4. Heuristic Clustering Evaluation Criteria	33
2.6. The Virtual Model	37
2.7. Visibility of Information	38
2.8. Classification as a Heuristic Process	39
2.9. The Problem-Solving Process as a Performance System	40

CHAPTER 3: A MODEL OF COGNITIVE SYSTEMS DEFINED FROM A PERSPECTIVE OF THE CLASSIFICATION PROCESS	43
3.1. Classification as a Bottleneck	43
3.2. Description of the Model	46
3.3. Implications of the Model	60
CHAPTER 4: KNOWLEDGE REPRESENTATION	62
4.1. Representation and the Cognitive Approach	62
4.1.1. Templates	62
4.1.2. Features	63
4.1.3. Structural Descriptions	65
4.1.4. First-Order and Second-Order Isomorphism	65
4.1.5. Prototypes	66
4.2. Annotated Predicate Calculus	67
4.3. The Chosen Representation	69
CHAPTER 5: CLASSIFICATION PROCESS	74
5.1. Components of the Classification Process	74
5.1.1. Relationships among the Components	74
5.1.2. An Alternative View of the Relationships among the Components	74
5.1.3. Comparison of the Views	77
5.2. Process Preparation	78
5.2.1. List of Goal-Relevant Attributes	79
5.2.2. Climbing a Domain Hierarchy	80
5.3. Clustering	83
5.3.1. Utilization of the Goal and Current Context	83
5.3.1.1. Evaluation of the List of Goal-Relevant Attributes	83
5.3.1.2. Best-First Search	85
5.3.1.3. Evaluation of the Resultant Clustering	90
5.3.1.4. GDN Rules and Threshold Update	90
5.3.2. Utilization of the Goal and Common Context	94
5.3.3. Utilization of the Associative Links	95
5.3.3.1. Evaluation of the Given Events' Associative Links	95
5.3.3.2. Associative Links Strength Update	96
5.4. Characterization	97
5.4.1. Generation of a Prototypical Description	97
5.4.2. Attribute-Value Relevance Calculation	99
5.4.3. Hierarchical Links Strength Calculation	100
5.4.4. Associative Links	101
5.4.5. Similarity Function	102
5.4.6. Other Approaches to Characterization	103
5.5. Building a Hierarchy	104

5.5.1. Description of the Implemented Approach	105
5.5.2. Other Approaches to Hierarchy Building	105
5.6. Feedback from the Performance System	106
CHAPTER 6: CLASSIFICATION ALGORITHM	109
6.1. Algorithm	109
6.1.1. The Outline of the Algorithm	109
6.1.2. Process Preparation	110
6.1.3. Clustering	111
6.1.4. Dialogue	112
6.1.5. Characterization	112
6.1.6. Building a Hierarchy	113
6.1.7. External Feedback and the Update Process	113
6.2. Algorithm Analysis	113
6.2.1. Properties	114
6.2.2. Knowledge-Base Flexibility and Adjustability	116
6.2.2.1. Incremental Learning	117
6.2.2.1.1. Concept Representation Update	117
6.2.2.1.2. Noise Handling	120
6.2.2.2. Retrieval	122
6.2.3. Efficiency Considerations	125
6.2.4. Semi-Automatic Generation of Event Representations	126
CHAPTER 7: IMPLEMENTATION	129
7.1. Domain of Application	129
7.2. Generated Code	130
7.3. Sample Runs	131
7.4. Summary	138
CHAPTER 8: CONCLUSION	139
8.1. Contributions of the Dissertation	139
8.2. Future Research Areas	142
APPENDIX	145
A: THE MAIN PROGRAM	145
B: THE UPDATE RULES	179
C: THE STRUCTURE OF THE ATTRIBUTE DOMAINS	182
D: ATTRIBUTES USED IN COMMAND DESCRIPTIONS	185
E: SELECTED COMMANDS	189
F: SAMPLE RUN 1: CURRENT CONTEXT	192
G: SAMPLE RUN 2: GIVEN CONSTRAINTS	210
H: SAMPLE RUN 3: GDN UPDATE	219

BIBLIOGRAPHY	231
--------------------	-----

LIST OF ILLUSTRATIONS

1.1	An example of the classification process	19
3.1	Homomorphism.....	45
3.2	Environment of the classification process	48
3.3	Working memory (WM).....	49
3.4	Knowledge base (KB)	52
3.5	Environment of the control process	52
3.6	Environment of the problem-solving process	53
3.7	Environment of the context-features extraction process	54
3.8	Environment of the expectations-generation process	55
3.9	Environment of the spreading-activation process	55
3.10	Environment of the encoding process	56
3.11	Environment of the matching and retrieval process	57
3.12	A model of cognitive systems viewed from the perspective of the classification process	58
5.1	Relationships among the components of the classification process	75
5.2	An outline of the implemented approach to the classification process.....	76
5.3	The <i>all-classes</i> approach to clustering	87
5.4	The <i>one-class</i> approach to clustering.....	88
5.5	An example of improved performance of the clustering process.....	94

LIST OF TABLES

1.1	8
5.1	102
6.1	124
7.1	134
7.2	137

ACKNOWLEDGMENTS

I wish to thank my advisor, David Y. Y. Yun, for his help and encouragement throughout my studies at Southern Methodist University. Many ideas in this work have emerged during discussions with him. Every time I have had a question, Dr. Yun has had an answer. His ability to attack a problem at the appropriate level of abstraction has always been a source of inspiration for me.

Many thanks go to Dr. Craig W. Thompson for his constant support throughout my work on the dissertation. He has made several valuable suggestions which have made this work more complete.

I thank Dr. Prasenjit Biswas, Dr Yu Hen Hu, and Dr. Branislav Meandzija, the other members of the supervisory committee, for their commitment to my work.

I wish to thank Dr. Robert Korfhage for providing me with helpful comments during the early part of this research.

I thank my parents, Zejfa and Sulejman, for letting me go so far away in my search for the best education when they needed me the most.

I wish to thank my children, Lejla and Adnan, for postponing my share of playing with them until after this work has been finished.

I thank my wife, Mirzeta, for her constant support and encouragement. I thank her for temporarily giving up many of her goals so I could complete this work, thus achieving my own. The dissertation, then, belongs to her as well.

The support of this research by the Department of Computer Science and Engineering at the Southern Methodist University, by the Institute of International Education under the Fulbright program, by the National Science Foundation under grant NSF MDR-84-70017, and by the Rudi Cajavec company is gratefully acknowledged.

CHAPTER 1

INTRODUCTION

1.1. Characterization of Classification

Classification is one of the basic mechanisms which provides humans with an ability to deal efficiently with the complexity of the real world. To be able to react properly to changes in the environment, people build a *mental model* of the real world, allowing them to infer the appropriate response and the expected state of the real world. Johnson-Laird [1983] has discussed the role of mental models in guiding human inference.

The inductive process of classification represents a mapping from the set of events in the real world to its mental representation. This process is a powerful data reduction tool used by cognitive systems to help in solving real problems. Very often the mapping takes the form of heuristic rules able to account, most of the time, for incomplete and partial information on input data as well as the influence of an ever changing environment.

At the same time, the mental model enables the classification process to reduce the concept description formation space by providing the goal and context of classification. It offers a framework for understanding the relationship between the classification and the problem-solving process, the latter being a performance system of the former. This internal feedback represents a source of constant dynamic changes in the system, which, in turn, adds to the adaptive power of a cognitive system and its ability to improve its performance with experience.

But, sometimes, the mental model does not provide the necessary environment for a successful classification. This is the case when the knowledge,

previously accumulated by the mental model, simply is not sufficient or adequate for solving the problem at hand. Moreover, it may be that the appropriate knowledge exists but has been stored in an inappropriate form that precludes the model to establish a relationship between the knowledge and the problem itself. In any of those cases, people may prefer to interact with the environment, seeking an additional source of information. This type of interaction represents the external feedback to the mental model and the classification process in particular.

It is the recognition of the importance of the discussed assumptions, the manner of their implementation in the classification algorithm, and the defined model of cognitive systems that represent the foundations of the approach presented in this work.

1.2. Machine Learning and Classification

Fisher and Langley [1985] have described classification as a process critical to the success of an intelligent organism. The ability to classify events (objects, states, observations, etc) as members of event families or classes is the basis of all inferential capacity. Work in Artificial Intelligence (AI) has concentrated significantly on developing methods for classification and the conceptual representations necessary to support these methods. More specifically, most of the work on classification has been conducted within the field of machine learning, under the common term *learning from observation* or *concept formation*. In order to understand the relationship between the work on classification and the field of machine learning, it is useful to characterize the research on machine learning.

Michalski [1986] has distinguished several basic learning strategies: *rote learning*, *learning by instruction*, *learning by deduction*, *learning by analogy*, and *learning by induction*. The latter subdivides into *learning from examples* and *learning by observation and discovery*. These strategies are ordered by the increasing complexity of the inference (transformation) from the information initially

provided to the knowledge ultimately acquired. In other words, this order reflects increasing effort on the part of the student and, correspondingly, decreasing effort on the part of the teacher.

1.2.1. Rote Learning

In rote learning the information from the teacher is more or less directly accepted and memorized by the *learner* (be that human or a computer program). There is basically no transformation. The main problem related to this type of learning is how to index the stored knowledge for future efficient retrieval. According to Carbonell, Michalski, and Mitchell [1983], variants of this method include:

- Learning by being programmed, constructed or modified by an external entity, requiring no effort on the part of the learner.
- Learning by memorization of given facts and data with no inferences drawn from the given information (the term “rote learning” is used primarily in this context).

1.2.2. Learning by Instruction

In learning by instruction (or *learning by being told*) the learner transforms the knowledge from the input language to an internally-usable representation. The new information must be integrated with already existing knowledge for effective use. The learner does perform some inference, but the main burden is still with the teacher (or other source) to present and organize knowledge in a way that incrementally augments the learner’s knowledge. Thus the basic transformations performed by a learner are *selection* and *reformulation* (mainly at a syntactic level) of information provided by the teacher.

1.2.3. Deductive Learning

This learning strategy was identified as a separate category only recently [Michalski, 1983] [Michalski, 1986]. Its characteristic is that the learner draws deductive, truth-preserving inferences from the knowledge given and stores conclusions proved to be useful. Examples of deductive learning are knowledge reformulation, knowledge compilation, creation of macro-operators, caching, chunking, equivalence-preserving operationalization, and other truth-preserving transformations.

1.2.4. Learning by Analogy

In learning by analogy the learner acquires new facts or skills by transforming and augmenting existing knowledge that reflects strong similarity to the desired new concept or skill into a form useful in the new situation. Learning by analogy combines deductive and inductive learning, where inductive learning characterizes the transformation process that involves generalization of input information and selection of the most plausible or desirable result. A common substructure to descriptions from different domains represents the basis for analogical mapping. Finding the common substructure is a form of induction, whereas performing analogical mapping has all the characteristics of deduction. Schank [1982] has defined *learning by being reminded*, which can be considered a form of learning by analogy. Carbonell [1983] has described learning by analogy through the process of formulating and generalizing plans from past experience.

1.2.5. Inductive Learning

Inductive learning is learning by generalizing facts and observations obtained from a teacher or environment. It can further be subdivided into *learning from examples* (also called *concept acquisition*) and *learning by observation and discovery*.

1.2.5.1. Learning from Examples

In this learning strategy, the task is to generate a general description of a concept that includes (explains) given positive examples and excludes all known negative examples of the target concept. Either a teacher or the environment represents a source of information providing the examples for the learner. The case when the source of information is the environment, on which the learner performs experiments and from which it receives feedback, is called *learning by experimentation*. Learning by experimentation includes *learning by doing* and *learning by problem solving*. *Stimulus-response learning* can as well be classified as a form of learning from examples.

There is a further subdivision within this form of learning: *instance-to-class* and *part-to-whole generalization*. When the learner is given independent instances of some class, and the goal is to induce a general description of the concept, it is an example of instance-to-class generalization¹. Most research done on learning from examples has been concerned with this type of generalization. Some results are discussed in the following references: [Kodratoff and Ganascia, 1986], [Lebowitz, 1986], [Quinlan, 1983], [Quinlan, 1986], [Sammur and Banerji, 1986], [Utgoff, 1986], [Vere, 1975], [Vere, 1978], [Winston, 1986]. A review of earlier methods for such generalizations is provided by Dietterich and Michalski [1983].

On the other hand, given selected parts of a whole event (object, scene, situation, process), the task of part-to-whole generalization is to hypothesize a description of the event. An example of this type of generalization would be learning to predict sequences, as described by Dietterich and Michalski [1983], where the task is to determine a rule (a theory) characterizing a sequence of

¹There is a distinction, exploited in this work, between the terms "concept" and "class". A "concept" is a general description (*intensional* definition) of a "class". A "class" is defined *extensionally* by listing all of its members.

objects or a process from seeing only a part of it.

1.2.5.2. Learning by Observation and Discovery

In learning by observation and discovery (also called both *descriptive generalization* and *unsupervised learning*), the learner searches for regularities and general rules/descriptions explaining all, or at least most, observations. It happens without the help of a teacher. The learner is required to perform more inference than in any approach discussed thus far. There are no instances of a specific concept provided, nor is there an oracle that can classify internally-generated instances as positive or negative ones. The observations may span several concepts to be acquired, rather than a single concept, thus introducing a focus-of-attention problem, as pointed out by Carbonell, Michalski, and Mitchell [1983].

Learning by observation and discovery includes: *constructing classifications*, *conceptual clustering* (a form of constructing classifications where the resulting classes are describable by simple concepts), fitting equations to data, discovering laws explaining a set of observations, and formulating theories that can account for the behavior of a system. *Genetic algorithms* [Holland, 1986] can be viewed as a variant of this learning strategy. Various research results in learning by observation and discovery are reported in the following references: [Lenat, 1983], [Amarel, 1986], [DeJong, 1986], [Langley *et al.*, 1986], [Stepp, 1984], and [Stepp and Michalski, 1986]. Lenat, for example, has designed a program, called AM, which demonstrates that new domains of knowledge can be developed mechanically by using heuristics. The second program developed by Lenat, Eurisco, has achieved some promising results in using heuristics to discover new heuristics as well.

1.2.6. The Components of Classification

Langley and Carbonell [1986] have specified some components or subprob-

lems which must be addressed by any system that learns from experience. Rather than trying to define learning in general, they have focused on the more constrained issue of learning from experience, as opposed to rote learning, learning by instruction, or deductive learning. According to them, the four basic components of learning from experience are:

- *Aggregation* - the learner must identify the events to be used for forming rules or hypotheses; i.e., the appropriate *part-of* relations must be determined.
- *Clustering* - the learner must identify which events should be grouped together into a class; i.e., the appropriate *instance-of* relations must be determined. This is, essentially, the process of generation of an *extensional* definition of the rule or hypothesis.
- *Characterization* - the learner must formulate a general description or hypothesis that characterizes instances of the rule. This is the process of generation of an *intensional* definition of the rule or hypothesis.
- *Storage/Indexing* - the characterization of the rule or hypothesis must be stored in some manner that lets one retrieve it when appropriate.

Based on these components of learning from experience, Langley and Carbonell have characterized the following five learning strategies: learning from examples, learning search heuristics, conceptual clustering, learning macro-operators, and grammar learning. The result is summarized in Table 1.1, originally published by Langley and Carbonell [1986, page 3].

On the other hand, Fisher and Langley [1985] have explained conceptual clustering processes as being composed of three distinct but inter-dependent sub-processes: the process of deriving a hierarchical classification scheme (not included in the approach presented above), the process of aggregating events into individual classes (clustering), and the process of assigning conceptual

descriptions to event classes (characterization).

Both approaches recognize the necessity of the clustering and characterization components (subprocesses). It is the third component where they differ. The first approach emphasizes the role of the *storage/indexing* component, while the second approach emphasizes the *hierarchy building* component. The storage component, however, is not unique to learning, but is rather common to all cognitive activities, and, as such, should not be treated exclusively within the research on learning. The hierarchy building component, on the other hand, is characteristic of classification systems, and should be addressed by the theories dealing with classification as a human cognitive activity. Even more, the *storage/indexing* component is a consequence of the *hierarchy building* component, in a sense that events and classes stored in a hierarchy can be used in classifying novel events.

Table 1.1

Relevant Components of Machine Learning Tasks	
Learning Task	Relevant Components
learning from examples	characterization
learning search heuristics	clustering, characterization
conceptual clustering	clustering, characterization, storage
learning macro-operators	aggregation
grammar learning	aggregation, clustering, characterization, storage

As a result, a view accepted in this work is the one of classification as a cognitive activity that consists of three (not necessarily independent) components (subprocesses): clustering, characterization, and building (constructing) a

hierarchy of concept descriptions. This view will be elaborated on within several chapters of this study.

1.2.7. Related Work in Machine Learning

The characterization component is shared by the classification task and learning from examples. Since there has been a substantial amount of work done on systems that learn from examples, the natural approach to building a classification system would be to solve the clustering problem first, and then to solve the characterization problem by employing one of the traditional methods of learning from examples. In fact, present conceptual clustering algorithms (the most popular method of classification) can be characterized in this way. A thorough analysis of these algorithms is given by Fisher and Langley [1985].

The GLAUBER algorithm [Langley *et al.*, 1986], concerned with discovering laws of qualitative structure in the domain of chemistry, forms classes based on the most commonly occurring relation (defined over an event set) and then characterizes these classes with respect to the remaining relations. The IPP algorithm [Lebowitz, 1983], dealing with generalization from the natural language text, constructs a number of alternative classes based on the *predictive* features (variable values) shared by all members of the class, and characterized by a conjunction of the *predictable* features shared by members of the class. The RUMMAGE algorithm [Fisher, 1984] needs a list of user-specified attributes to form clusterings over a set of given events. Each clustering is implied by the values of a distinct attribute, and the clustering with the “best” conceptual description of given events over the remaining attributes gets selected. Thus, RUMMAGE solves the clustering problem by using individual attribute values to imply possible classes and then employs a learning from examples algorithm to characterize classes in terms of the remaining attributes. This method is then applied recursively to each of the resulting classes, thus effectively constructing a hierarchical

classification scheme. The DISCON algorithm [Langley and Sage, 1984] uses the same strategy to cluster a set of given events as RUMMAGE does. But, unlike RUMMAGE, it does not generate an explicit description of the resulting classes over the remaining attributes. It, rather, simply calls itself recursively on each of the resulting classes, thus forming a classification tree over the events of each class with respect to the remaining attributes. Both algorithms are, however, based on Quinlan's ID3 program for learning from examples.

CLUSTER/2 system [Michalski and Stepp, 1983] [Stepp, 1984] solves the clustering problem differently than the systems described above. It selects N *positive instances*, if N disjoint classes of given events is to be generated, by choosing N *seed* events (initially done randomly). Each seed is regarded as a positive instance of some class, and all other seeds as negative instances of the same class. The program then derives *maximally-general discriminant descriptions* for each class implied by the seeds, in such a way that each description covers only the positive instance but no other seed. This process ensures that there is at least one cluster covering an arbitrary event. Once all seed and nonseed events have been classified with respect to the maximally-general discriminant descriptions, *maximally-specific characteristic descriptions* are defined for each class. These descriptions effectively reduce the possibility of overlapping clusters with respect to unobserved events. The system then selects one description for each seed and evaluates the resulting clusters (made disjoint by the special-purpose procedure) according to a prespecified heuristic criterion. If for the given number of iterations the system does not produce a better clustering, the system stops searching for a better solution.

Stepp and Michalski [1986] have recognized the importance of both the goal of classification and the existence of background, domain-specific knowledge. Their role in reducing the hypotheses-formation search space and increasing the

psychological plausibility of the resulting classification is extremely valuable.

As Fisher and Langley [1985] pointed out, conceptual clustering methods may be viewed as extensions of techniques of *numerical taxonomy*, a collection of methods developed by social and natural scientists for creating classification schemes over event sets. The quality of clustering, in numerical taxonomy methods, is a function only of generated classes. The quality of clustering is dependent neither on the quality of concepts which may be used to characterize clusters of the clustering nor the map between concepts and the clusters they cover. The resultant clusters may not be well characterized in some conceptual language understood by humans, which precludes numerical taxonomy methods from use in systems built to match human performance in building classifications.

There are many research directions that deal, more or less, with issues relevant to the classification process.

Watanabe [1985] argues that similarity (the bond by which instances of a concept - members of a class - are supposed to cohere) depends on the goal of classification, and that classification depends on the concept of similarity. To avoid this cyclic dependence, the quality of classification should be evaluated by its utility (usefulness) in the course of the problem-solving process. The concept of similarity is, then, indirectly evaluated through it.

Mitchell and Keller [1983] have used background knowledge to guide an inductive learning program for acquiring problem-solving heuristics in integral calculus.

Vere's THOTH system [Vere, 1978] represents a possible basis for a conceptual clustering system for structured objects. THOTH discovers a minimal set of generalizations which cover a given set of relational production instances. The resulting data-driven hierarchical classification resembles, in many ways, a result

of a bottom-up (agglomerative) approach to conceptual clustering.

Lenat [Lenat, 1983] has demonstrated that new domains of knowledge can be developed mechanically by using heuristics. As new heuristics are needed, with emerging new domain concepts, they in turn can be discovered by using a body of heuristics for guidance. Thus, using heuristics to guide a learning process represents a powerful and simple strategy which helps to bring flexibility and generality into learning algorithms.

Classifier systems of Holland [1986] operate with highly general learning mechanisms applied to a simple representational scheme. They represent a class of message-passing, rule-based systems, in which a large number of rules can be active simultaneously. Individual rules are kept simple and standardized as a result of the strategy that combinations of rules are used to define complex situations. The system gains flexibility, and the objective becomes that of finding rules that serve well in a variety of tasks. Default hierarchies are easy to generate and use. Rules can be tied together into networks of various kinds by appropriate use of tagging. The most difficult inductive task is that of generating plausible new rules. The task is carried out by a *genetic algorithm*. It uses high-strength classifiers as the generators of new classifiers. Systems based on these principles have been tested in a variety of contexts: poker playing, gas pipeline transmission, etc.

Prieto-Diaz and Freeman [1987] have attacked the interesting problem of locating, retrieving, and reusing software components from a large collection. They have proposed a *faceted classification scheme* based on reusability-related attributes and a selection mechanism as a partial solution to that problem.

Some authors ([Chandrasekaran, 1986], [Clancey, 1985], and [Hadžikadić, Yun, and Ho, 1986]) have considered classification from a different perspective. In their search for an alternative level of abstraction of the information processing

task, they have discussed the role of classification within a problem-solving process that would help in system design, knowledge acquisition, and explanation [Chandrasekaran, 1986]. Clancey [1985] has defined *heuristic classification* as the method that systematically relates data to a pre-enumerated set of solutions by abstraction, heuristic association, and refinement. Hadžikadić, Yun, and Ho [1986] recognized the importance of the model of cognitive systems for classification and problem solving in general. However, the research presented in this work is concerned rather with the classification process itself and the set of constraints imposed on it by the problem-solving process.

1.3. Related Work in Psychology

Research on classification processes within AI has gained significant benefits from the research on categorizations within cognitive psychology, and vice versa. This section reviews some of the research efforts that have considerably influenced the research in the field of machine learning.

Rosch, Mervis *et al.* [Rosch, 1973] [Rosch and Mervis, 1975] [Rosch *et al.*, 1976] [Rosch, 1978] have hypothesized that the members of categories which are considered most *prototypical* are those with the most attributes in common with other members of the category and the least attributes in common with other categories. The instances of categories fall on a continuum from prototypical instances to unclear borderline cases. A prototype is neither necessary nor sufficient to represent all that is induced about the category structure. The hypothesis is that, in probabilistic terms, prototypicality is a function of the *total cue validity* of the attributes of items. Rosch and Mervis have argued that *family resemblance* offers an alternative to criterial features in defining categories.

Tversky and Gati [1978] have analyzed a concept of similarity and similarity relations. They concluded that there is no unitary concept of similarity that is applicable to different domains and situations. Rather, it appears that there is a

wide variety of similarity relations (defined on the same domain) that differ in the weights attached to the various arguments of the *feature matching* function. Tversky and Gati admit that similarity is relative and variable - events can be viewed as either similar or different depending on the context and frame of reference. Also, similarity often does not account for our inductive practice but rather is inferred from it. However, they believe that these facts only point out the need for a comprehensive theory that describes not only how similarity is assessed in a given situation but also how it varies with a change of context.

Murphy and Medin [1985] have argued for a theory-based rather than a similarity-based approach to conceptual coherence. Conceptual coherence is viewed to be derived from the position of a concept in the complete knowledge base. Concepts are coherent to the extent that they fit people's theories about the world.

Barsalou's research on goal-directed categories [Barsalou, 1981] [Barsalou, 1983] provides illustration of the influence a context can exert on conceptual structure. Knowing the goal that defines the category greatly facilitates category learning, and the typicality structure of goal-directed categories is determined by how well the exemplars satisfy the goal rather than by family resemblance [Barsalou, 1981]. Also, Barsalou found evidence suggesting that natural concepts may be organized around dimensions that reflect the relationship between the concept and broader goals and knowledge.

Wattenmaker *et al.* [1986] have suggested that, in order to fully understand the constraints on category formation, it is necessary to focus on the interaction between the types of encodings people find natural and the structure of the environment. They pointed out that the practice of dividing concepts into their constituent parts has led to the tendency to view categories as little more than a collection of features and categorization as simply a process of attribute match-

ing.

Collins and Loftus [1975] have defined the *spreading activation* process that may contribute significantly to the psychological plausibility of derived classification. The activation spreads out from processed concepts (events, instances). When a concept is accessed, its associative links get activated, thus activating descriptions of events non-hierarchically associated with the accessed (source) concept. Every associative link has attached a corresponding *strength* to it. If there are concepts non-hierarchically linked to more than one source concept, they receive the combined activation strength of their sources. The concepts scoring above the predefined threshold will get a chance to make a contribution to the classification process. Thus, along with the goal and context of classification, the spreading activation process contributes to the psychological plausibility of the classification process. Similarly to categorical links of concepts, these associative links will be updated according to their contribution to the success or failure of the current classification process.

Classification (and induction in general) depends on the ability to accommodate variability in the environment. Fried and Holyoak [1984] have indicated that people use *variability* information to classify novel instances. Holland *et al.* [1986] have discussed that the dispersion of concept instances over their dimensions of variations is not simply error variance in its usual sense, but is itself a property of the environment. In order to classify events into categories, people need not only knowledge of central tendencies (if there were overlap among the properties of the category alternatives), but also some estimate of the dispersion of the dimensions defining each category before classification could be justified. The same knowledge, then, could be used for a decision when a new concept should be formed, given a set of events.

Anderson [1983] has defined ACT* system - a theory of *cognitive architec-*

ture. It is a theory of the basic principles of operation built into the cognitive system. ACT* is concerned with higher-level cognition or thought. It is assumed that higher-level cognition constitutes a unitary human system. Anderson claims that a central issue in higher-level cognition is control - what gives thought its direction, and what controls the transition from thought to thought. His major concern has been to understand the principles behind the control of thought in a way that exposes the adaptive function of these principles.

1.4. Approach Taken in this Work

Research in machine learning encompasses three interconnected orientations [Michalski, 1986]:

- Theoretical analysis and development of general learning algorithms.
- The development of computational models of human learning processes (*cognitive modeling*).
- Task-oriented studies concerned with building learning systems for specific applications.

Research in the first orientation tries to develop algorithms that solve theoretical learning tasks independent of application. There is no attempt to develop an algorithm that is similar to the one a human might use to perform the given task. Human learning, however, is the focus in the second orientation. Its goal is the development of computational theories and experimental models of human learning. Contrary to these two orientations, the third one, an *engineering* orientation, deals with specific practical learning tasks and tries to develop engineering systems capable of performing these tasks. Useful ideas from the other two orientations are readily adopted in this orientation. At the same time, a solution to a specific problem can be generalized to solve a whole class of similar problems.

In July 1980, Herbert Simon [Simon, 1983] delivered a controversial keynote address at the Carnegie-Mellon Machine Learning Workshop. He concluded that, with the exception of cognitive modeling, some rethinking of long-term objectives was in order. Simon defined five priorities for learning research. The first two of them are defined on page 35 of his article [Simon, 1983]:

1. I would give a very high priority to research aimed at simulating, and thereby understanding, human learning. It may be objected that such research is not AI but cognitive psychology or cognitive science or something else. I don't really care what it is called; it is of the greatest importance that we deepen our understanding of human learning, and the AI community possesses a large share of the talent that can advance us toward this goal.
2. I would give a high priority, also, to basic research aimed at understanding why human learning is so slow and inefficient, and correspondingly, at examining the possibility that machine learning schemes can be devised that will avoid, for machines as well as people, some of the tediousness of learning.

It was the same line of reasoning that motivated this attempt to define a model of cognitive systems, developed from the classification component point of view, in order to place classification, as a cognitive process, in its natural environment. None of the research efforts discussed previously in this chapter has attempted to approach the problem integrally, taking into consideration the cognitive nature of the classification process. Consequently, those research efforts are characterized by a lack of a performance system serving as a source of internal feedback.

As a result, the following sequence of steps is accepted in this work:

- develop a model of cognitive systems from a perspective of the classification process,
- understand the nature and form of relationships between the classification component and the rest of the system,

- extract a set of constraints imposed by the system on the classification process,
- define a classification algorithm.

1.5. Results of the Dissertation

It is useful for the further discussion to state at the beginning the main results of the research described in this study:

- a model of cognitive systems is defined from a perspective of the classification process;
- a classification algorithm is described, implemented, and evaluated in the domain of the UNIX user commands;
- two levels of feedback are defined;
- the learning mechanism for updating the declarative and procedural knowledge used in the classification process is implemented;
- the method for the incremental update of the concept description is outlined;
- the problem of noise handling is significantly reduced.

1.6. An Example

The example presented in figure 1.1 explains the process of classification and the end result in the domain of the UNIX user commands.

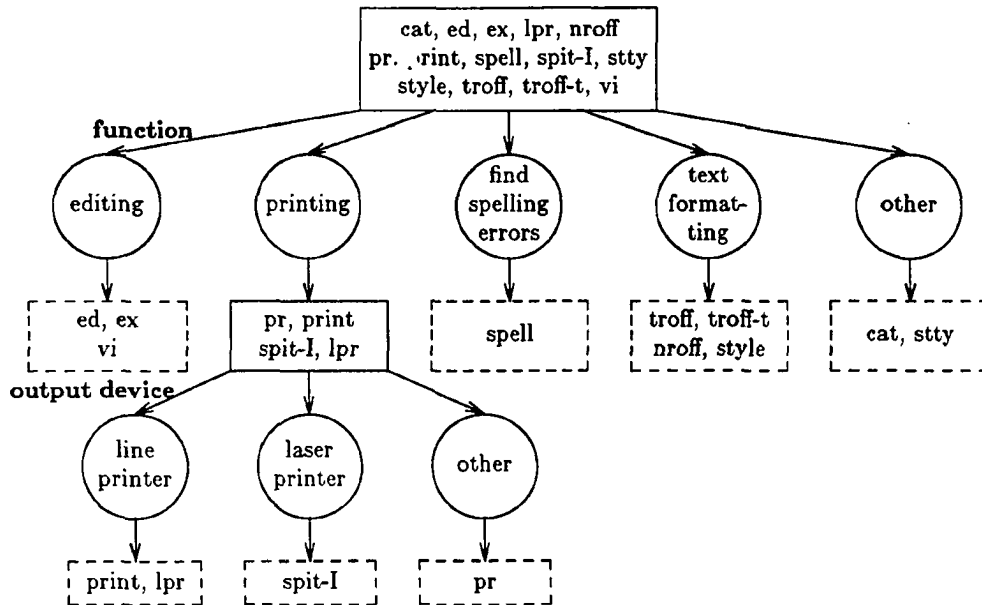


Figure 1.1 An example of the classification process.

Given the set of the events to be classified, the task is to distinguish classes $C_1, C_2, \dots, C_n$, such that the resulting classification is of high quality with respect to a predefined criterion. The first problem is to find the appropriate dimensions (attributes) for dividing a class into subclasses. The second problem is to find the class to be evaluated next. The third problem is to define the stopping criterion.

In the example, the attributes **function** and **output device** are used as the dimensions for classification. The events are grouped according to their values (written inside a circle) with respect to these attributes. The number of generated classes is 7 (dashed boxes). All the commands that belong to the same class are equivalent with respect to the properties used in the derivation of the class. The same figure is used in different context in Chapter 5.

1.7. Outline of the Dissertation

The research described in this study deals with problems of concept formation by classification. In the chapters that follow, the classification process, the mental model, and their relationship will be analyzed in detail. A classification algorithm that reflects the nature of that relationship will be outlined.

Chapter 2 will lay out the foundations of this research. It characterizes relevant aspects of the mental model as well as kinds of background knowledge needed for successful implementation of the algorithm. Chapter 3 describes the model of cognitive systems from the classification component point of view and its implications for the process of classification. Chapter 4 specifies the knowledge representation mechanism adopted in this work and a justification for that decision. The classification component itself is described in Chapter 5 with a detailed description of its subcomponents. Chapter 6 contains a specification of the algorithm. First, a general version of the algorithm is given, but then each subprocess is explained in detail. An analysis of the algorithm is presented at the end of the chapter. Various aspects of implementation of the algorithm as well as results and examples of sample runs are described in Chapter 7.

CHAPTER 2

RESEARCH FOUNDATIONS

2.1. A Goal of the Research

As pointed out in the previous chapter, a goal of this research was to define an approach to classifying a set of given events in a manner that guarantees a *useful* final result. A result is considered to be useful if it is intelligible to people and can be readily used for a solution of the current problem-solving task.

Another goal of this research was to outline a system that would demonstrate an ability to adapt easily to a novel situation, a system that would improve its performance with experience. These abilities are of critical importance for the survival and success of a cognitive system. For that to happen, the system must be able to update (modify, improve) dynamically its existing knowledge. A proper and efficient mechanism, heuristic in nature, that would serve as a driving force of the modification is in order.

This research has been primarily concerned with classification of *unstructured* events. The description of such events involves attributes of events as a whole. The view accepted in this work suggests that the problem of classification should be addressed first in the context of unstructured events and, consequently, at the wholistic level. Research in psychology has shown that people make use of the information on structural components of events only if they fail to classify events at the wholistic level. Since the goal of this work has been to produce a psychologically plausible classification of the set of given events, it was a justified direction in which to proceed.

2.2. Classification as a Model-Based Process

Classification is a cognitive activity and as such does not stand alone within a cognitive system. It interacts constantly with the rest of the system. Sometimes it needs specific information, while at some other time it may need an action to be performed by some other cognitive process. At the same time, the classification process may help other cognitive processes to perform their functions effectively. The quality of performance of any component of a cognitive system depends not only on the quality of input data and the component itself, but on the quality of interaction with the rest of the system as well. The same is true for the classification component.

But what kinds of information does it need from the other components? Although this issue is going to be treated in more detail in the next chapter, an initial discussion is presented here. The first kind of information it may need is a *goal* of classification along with a *current* context. This information represents a focusing mechanism on the properties of the environment that are of crucial importance to the learner. These properties will ensure the usefulness of the result of classification to the system with respect to its goals and intentions.

The second kind of information is a *common* context. Very often people are forced to deal with incomplete information. If it is a novel situation, we have a problem. But, if it is a situation similar to the one(s) that we have already had a chance to deal with, maybe we can give rise to some plausible expectations that would help us make the existing information as complete as possible. That would allow the system to continue drawing inferences consistent both with the goal and the given information.

The third kind of information that could be supplied to the classification component by the rest of the system is the set of events *associatively* related to the events to be classified. This observation is based on the work done by Collins

and Loftus [1975] on the *spreading activation* process (described in greater detail in Chapter 1), which may contribute significantly to the psychological plausibility of classification. But when is this important? This kind of information may play an important role in novel situations. In those situations a common context would not help much because the system has not had an opportunity to build one that is at least partially suited to the current context. The only existing information related to the events to be classified is, then, a set of events that the system is reminded of after having seen the input events. If a specific event gets pointed to by several input events, it must be that some of its properties are shared by (or important to) those events. Its properties, then, may be tried as a substitute for the information that could be inferred with the help of the common context. The same line of reasoning may work as well in situations that are not novel but where the corresponding common context is simply not enough to ensure a plausible classification.

Which cognitive processes are responsible for providing such information? What kind of declarative and procedural knowledge, and in what form, is available to which process? What is the underlying *structure* that makes it all possible? What makes a human so efficient, flexible, and adaptive? It can be argued that the answer to these questions is given in the form of a *mental model*, along with the underlying architecture of a cognitive system. People build a mental model of the outside world in order to deal efficiently with its complexity. It grows with experience. In fact, it is the embodiment of that experience. The more we know, the better (the more complete) the mental model. At the same time, a *virtual model* describes the current situation. It contains a description of the immediate environment as well as the knowledge brought to deal with it. However, none of this would be possible without a flexible and efficient underlying organization (architecture, environment) offered by the cognitive system as a

whole, to which both mental and virtual model belong as well.

In conclusion, it is the mental model and internal organization of a cognitive system (to be described in Chapter 3) that provide the environment for interactions between different cognitive processes and the help (additional information) they give to each other. It is the virtual model that comprises both a description of the current challenge coming from the outside world and the set of procedures and declarative knowledge, provided by the mental model, brought together to deal with it.

2.3. Classification as a Goal-Driven Process

Michalski and Stepp [Stepp and Michalski, 1986] [Stepp, 1984] have pointed out a need for supplying the system with a general goal of classification. It has helped them to avoid the necessity of defining explicitly relevant *descriptors*² and inference rules for deriving *new* descriptors. A goal of classification exists prior to initiation of the process. It is always the goal that is specified first. The goal is either supplied externally by the environment, or generated internally by the problem-solving process. Whatever the reason, the goal is there and it is up to the classification process to use it properly.

What are the benefits of knowing the goal of classification? The most important one is the improved *quality* of the result of classification. Here, a quality of the result is defined as a function of both its *understandability* by people (or a cognitive system for that matter) and its *usability* for the problem-solving task waiting for that result. The better the understandability and/or the greater the usability, the higher the quality of the result. But understandability and usability are not mutually independent. The understandability improves the usability of the result of classification. Humans use information in a more appropriate and

²In the terminology that Michalski and Stepp use, descriptors include attributes, *n*-ary functions, and relations used to characterize events.

flexible way when they understand it; and, vice versa, if they can use it properly, the underlying assumption is that they understand at least some relevant aspects (properties) of it.

The second benefit of having a goal of classification is its potential to reduce drastically the *hypotheses* search space, where *hypothesis* denotes a description of the class. The search space is the power set of the set of all (*attribute*, *value*) pairs used in the description of events to be classified, plus the (*attribute*, *value*) pairs that can be inferred from them through the process of *generalization*. So, if the cardinality of the set is n , the size of the search space is 2^n . Thus, reducing the number of (*attribute*, *value*) pairs in the set reduces significantly the search effort. How can a goal of classification help the system to reduce the number of (*attribute*, *value*) pairs? It helps the system to effectively distinguish between the *relevant* and not-so-relevant (*attribute*, *value*) pairs, just by explicitly providing a *bias* of the classification process. A (*attribute*, *value*) pair is relevant if it is relevant to the goal of classification. It may even be possible to define the *degree of relevancy* property, and define it as a number between 0 and 1, with 0 assigned to the (*attribute*, *value*) pair *not* relevant to the goal and 1 assigned to those pairs that are *absolutely* relevant to the goal of classification.

How does the system know which (*attribute*, *value*) pairs are relevant to which *goal* in which immediate *context*? For that purpose, Michalski and Stepp's idea of a *Goal Dependency Network* (GDN), which organizes goals and goal-relevant descriptors into a network, is modified to account for modifications caused by the change of context and to include the information on the degree of relevancy between the goal and specific descriptor. The idea of modified GDN will be explained in greater detail later in this chapter.

2.4. Classification Combining Model- and Data-Driven Processes

There are, generally speaking, two ways of hypothesizing a description of the class:

- *model-driven* - the hypotheses are generated by the model and tested by data,
- *data-driven* - the hypotheses are generated on the bases of data.

It is the nature of the adopted approach to classification that will point out the right choice between the two.

As pointed out in Chapter 1, there are three subprocesses of the classification process: clustering, characterization, and a hierarchy building. Each of them can be characterized separately with respect to the nature of the control mechanism. The clustering process, as discussed in the previous section, is heavily dependent on the goal of classification. If we are to implement the possibility of different views of the same data, then we should not regard that data as responsible for clustering. The data will certainly modify the list of (attribute, value) pairs relevant to the goal of classification (what happens if there is no data with the specified property?), but will not drive the process of clustering. (The list of goal-relevant attribute-value pairs will modify event descriptions as well, according to the structure of attribute domains). Also, since there is a degree of relevancy of the (attribute, value) pairs to the goal, it is possible to list them in decreasing order according to that degree. What that means is that the system can cluster the whole set of events according to their values for the attribute with the (attribute, value) pair on the top of the list. The newly generated clusters can be further subdivided into sub-clusters with respect to the next attribute in the list and so on, thus effectively building a tree-like structure with discriminating (attribute, value) pairs as labels on the links (arcs) and resulting classes as leaves. Consequently, it may be concluded that the clustering is a top-down,

context-and-goal-dependent, model-driven process.

Once the set of classes is generated, the next task is to generate a description of each of them. One possibility is to use the *conjunction* of the (attribute, value) pairs on the path to the specific class. These pairs must be contained in the description of each and every event in that class, otherwise the event would not be there. However, it is possible to capture more information in the class description, with no significant additional computation cost, by including all of the (attribute, value) pairs found in descriptions of at least half (or so) of the events in the class. Certainly, there is a counter-question: what is it to be gained by doing so? The following are possible benefits:

- a) additional (attribute, value) pairs will create fuzzy boundaries of a concept, since they may be shared by events from other classes, thus effectively implementing a concept of intersecting categories;
- b) those pairs will add to the information (and consequently an explanation) power of the concept, since it will already possess (attribute, value) pairs that may become important in another context;
- c) they are true of a number of events in the class (above the pre-set threshold), and should be included in its description.

Since the (attribute, value) pairs used in concept formation are those used in descriptions of given events, the characterization has all the properties of a data-driven process.

Finally, a hierarchy building process resembles the characterization process. It uses leaf concepts, generated by the characterization process, as *new events* to be characterized. This process is then repeated iteratively, until the root of the tree is reached. As a result, hierarchy building is a bottom-up data-driven process.

The important difference between the approach implemented here and *similarity-driven* (similarity-based) approaches is contained in the fact that whether two events are similar or not is defined (in this approach) by both the mental and virtual model, rather than according to a predefined, mostly syntactic, similarity function. Similarity is a context dependent property, and is rather a consequence of classification as an inductive process. In Chapter 6, the retrieval process and the fuzzy concept of similarity will be discussed.

2.5. Background Knowledge

In order to create a *meaningful* classification, a cognitive system must possess *background knowledge*, which includes a goal-dependency network, associative links between events and/or concepts, a description of the structure of attribute domains, and heuristic clustering-evaluation criteria.

2.5.1. Goal-Dependency Network

GDN relates goals, common context, and relevant (attribute, value) pairs. It can be implemented as a series of rules in the following format:

if *goals* and *common context*
then *(attribute, value) pair*
 with the *strength*

GDN is a result of our experience. At the beginning, with no help from GDN, the system produces meaningless classifications, much like a child experiencing the surrounding world with no previous knowledge of it. The more the child spends time in that environment and learns about it, the better the groupings of *similar* objects he/she creates. GDN is generated by many processes, one of which is a classification itself. All kinds of learning participate in that process of generating GDN, such as learning by being told, learning from examples, learning by anal-

ogy, learning by discovery and observations, etc., as well as processes of a different nature, such as a problem-solving process, doing experimentations, and so on. Consequently, a cognitive system must provide feedback to GDN from processes that use its content in the course of the action they perform. If the information from GDN proves to be useful, that must be reflected in the increased *strength* of the corresponding rule(s). Otherwise, the strength should be decreased accordingly. As a result, the strength reflects the usefulness of the rule in the past. The same discussion certainly holds for the classification process as well.

An example of rules from GDN is presented below. Since a domain of UNIX user commands is chosen to test the classification algorithm, the example is drawn from that domain. So, for instance, if the goal was to write a cover letter for an application for a job, then the following rules could fire:

```
(rule #1
  if
    (goal write-a-letter)
    (context
      (domain UNIX-user-commands))
  then
    (relevant-attribute-value-pair
      (function editing)
      (strength 1.0)))

(rule #2
  if
    (goal write-a-letter)
    (context
      (domain UNIX-user-commands)
      (type-of-letter cover-letter)
      (recipient-of letter official-person)
      (reason-for write-a-letter job-search))
  then
    (relevant-attribute-value-pair
      (function text-formating)
      (strength 0.8)))
```

```

(rule # 3
  if
    (goal write-a-letter
     (context
      (domain UNIX-user-commands)
      (recipient-of letter official-person)
      (reason-for write-a-letter job-search)))
  then
    (relevant-attribute-value-pair
     (function find-spelling-errors)
     (strength 0.85)))

```

```

(rule # 4
  if
    (goal write-a-letter
     (context
      (domain UNIX-user-commands)
      (type-of letter cover-letter)
      (recipient-of letter official-person)))
  then
    (relevant-attribute-value-pair
     (output-device laser-printer)
     (strength 0.7)))

```

All of the above rules would fire if the context was described by the following (attribute, value) pairs:

```

(domain UNIX-user-commands)
(type-of letter cover-letter)
(recipient-of letter official-person)
(reason-for write-a-letter job-search)

```

Let us notice that the *domain* predicate in the description of GDN rules gives the system a chance of maintaining a description of several application domains within a single knowledge base, effectively serving as a filter for the rules of a specific domain when needed.

GDN allows the system to utilize the difference between the *current* and *common* contexts. The goal and current context of classification select among the GDN rules the ones that will fire. If the (attribute, value) pairs generated by firing the corresponding rules are sufficient for successful clustering, the process is

over. Otherwise, the list of goal-relevant (attribute, value) pairs should be extended. One way of doing that is by utilizing a concept of common context. If the system is able to come up with predictions of what else may be expected in the current situation, based on past experience, then more GDN rules will be ready to fire, thus adding more (attribute, value) pairs to the list of goal-relevant attributes, which, in turn, may help the system to come up with a successful clustering. This would not be an unusual situation because of the fact that people tend to describe a problem rather incompletely, leaving a substantial amount of information to be communicated through the so-called common-sense knowledge. Later on, when the model of cognitive systems is described, two processes, namely *context-features extraction* and *expectations-generation*, will be defined as responsible for *filling-in-the-blanks* when faced with an incompletely described environment.

2.5.2. Associative Links

The other way of extending the list of (attribute, value) pairs to be used as a basis of classification is to use the result of the spreading information process, i.e. the activated associative links of the events to be classified. The event that earns a cumulative support from the events to be classified, with the score above the pre-set threshold, may delegate the (attribute, value) pairs used in its description to the list. The rationale behind this observation is that it was those attributes which made the events related to each other, and it is certainly likely that they may help classification of given events in a meaningful manner. The associative links allowed by the system are *event - event*, *event - concept*, *concept - concept*. Of course, the strength of the associative links should be updated dynamically. The links that pulled out the events with the (attribute, value) pairs that proved to be useful in the context of generating a classification should have the strength increased accordingly. Otherwise, the strength of those links should be decreased.

Several authors have recognized the importance of the concept of spreading activation. Anderson [1983] postulates a process of spreading activation through the semantic network that is independent of the execution of rules. However, the assumption that spreading activation proceeds automatically is not unique to Anderson's theory. As pointed out earlier, Collins and Loftus [1975] have made spreading activation a central element of their theory of the retrieval of conceptual knowledge. Spreading activation typically is assumed to have four properties: (1) when a concept is activated in memory, activation spreads to the concepts directly associated with it semantically; (2) the process requires no processing capacity and hence will not suffer interference from any concurrent cognitive process; (3) activation continues spreading from the initial associates to their associates, and so on indefinitely; and (4) the entire process is extremely rapid (almost instantaneous).

The evidence for the first property of spreading activation has been shown rather strongly in various reports on the research done in psychology. However, there is no substantial evidence for the other three properties to hold. In fact, significant evidence has been provided, by several authors, against *automatic* spreading activation. Holland and his colleagues [Holland *et al.*, 1986] have argued that the spread of activation from concept to concept is under the control of rules, their corresponding *support levels*, and their *competitive interactions*. In order to be computationally feasible, the system that assumes the *rule-directed* spread of activation must be built on the notion of parallel execution of rules. In this work on classification, the rule-directed approach to spreading-activation is adopted.

2.5.3. The Structure of Attribute Domains

The knowledge of the structure of attribute domains is extremely important to the classification process. It is the richness of the structure of attribute

domains that makes the hypotheses search space so immense. If the structure of an attribute domain is a hierarchical one, the classification algorithm will not take into account the leaf values only, for the resulting classification would not be at the appropriate level of abstraction as requested by the problem-solving process or the end user, and not useful as such. But, then, which one out of many intermediate domain values at the different levels of abstraction should the system choose as the one that gives the most useful result. What level of abstraction is the *right* one?

The decision has all elements of a heuristic, for there is no generally agreed *best* classification. It depends on the goal of classification, context, and the current amount of knowledge. The heuristic that sounds plausible is the following one: *the goal and the context of the problem will define the appropriate level of abstraction of the attribute domain to be used as a basis of classification.* (This heuristic will be discussed in greater detail in Chapter 5). In other words, it says that the system should try to infer (attribute, value) pairs from the list of goal-relevant attributes in the description of events to be classified. For that to happen, the system should possess a set of inference rules capable of inferring the appropriate (attribute, value) pairs in the description of events, given the list of goal-relevant attributes. This is a top-down, model-driven process, and, as such, reduces the search process significantly, introducing, on the other hand, a possibility that the optimal solution will not be found, if there is any at all.

2.5.4. Heuristic Clustering Evaluation Criteria

Michalski and Stepp [Michalski and Stepp, 1983] [Stepp, 1984] [Stepp and Michalski, 1986] have defined Lexicographical Evaluation Functional with tolerances (LEF) as a heuristic evaluation criterion of the quality of generated clustering. It is comprised of several criteria specifying desirable properties of a classification. Each elementary criterion measures a certain aspect of generated

classification, such as the relevance of descriptors used in the class descriptions to the goal of classification, the fit between the classification and the objects, the simplicity of the class descriptions, the number of attributes that singly discriminate among all classes, and the number of attributes necessary to classify the objects into the proposed classes. The user specifies an order of application of these criteria to the result of classification. The clusterings that score equally (within the tolerances) on the top-ranked criterion keep on going. At the end, the system may run out of either criteria or candidate clusterings. Finally, if there are two or more clusterings that survive this process, the winner is chosen randomly.

There are several problems with LEF. It is applied at the end of the process to compare the result of the last iteration with the best one of the past iterations. When does the process stop? It stops when there is no improvement in the predefined number of iterations. How do we define that number? What are the parameters of that decision? Also, why is it that we have to wait for the end of the characterization process to evaluate the result of classification? How does the characterization change the quality of the result of the clustering process? If that is the case, isn't it true that the representational mechanism is not the most appropriate one? There are several problems with the elementary criteria as well. The relevance of descriptors used in the class descriptions to the goal of classification should drive the classification process rather than serve to evaluate its result, once the classification is over. The fit between the classification and events is not a criterion that people really use so often. Do we care what the maximal number of events possibly covered by the description of the class is? Do we care what percentage of the maximal number of events are there in the set of events to be classified? The answer to both questions is *no*. Also, the simplicity of class descriptions is hardly an objective criterion. What is simple in one

language may not be simple in another. Isn't, then, the simplicity a characteristic of the chosen representational mechanism rather than the classification itself?

The approach presented here points out that the evaluation criterion should be applied at the end of the clustering process. Once the system is satisfied with the result of clustering, the characterization of generated classes should be pretty straightforward, provided the structure of attribute domains has already been taken into account in the clustering phase. Also, according to the model defined in this work, if successful, the first clustering must be the best one, since the goal and current context of classification have been consulted. However, if it is not successful, the system will first take into account the common context and then, if necessary, the associative links. And that is it. What happens if there is no successful clustering after all? Well, what do people do? They consult the immediate environment, conduct experiments, ask, etc. That is exactly what the system can do. It can consult the user and use his/her response accordingly (learning by being told).

But, what is the heuristic clustering-evaluation criterion that is going to be applied? Similarly to Michalski and Stepp's approach, there may be more than one elementary criterion. How many and what they are, should be answered by the research in cognitive psychology as well as cognitive science. This study has identified one of them: *a distribution of events across the generated classes should be roughly uniform*. The rationale behind this criterion was that, given ten events to be classified, people are hesitant to suggest three classes, for example, with one event in the first, one event in the second, and eight events in the third class. This was the criterion that was used in the implementation and testing phase of the algorithm. The number of classes to be generated may be predefined by the user or problem-solving process as well. Although that criterion cannot be accepted as a heuristic one, the algorithm should be capable of handling this case

accordingly.

The heuristic clustering-evaluation criterion should have a dynamic structure rather than a static one. Consequently, each elementary criterion should have a *strength* associated with it, reflecting its usefulness in the past. Only the elementary criteria with the strength above the pre-set threshold participate in the heuristic clustering-evaluation criterion. Generated clusterings have to satisfy all of the elementary criteria. However, the system must give a chance to the elementary criteria with the strength below the threshold to prove themselves, should the need arise. For that to happen as well as to enforce the adaptability and flexibility of the system, the system must provide the feedback to the classification component as to whether it was successful with respect to the goal of classification or not. The strength of the participating elementary criteria should be changed accordingly. If there are no mistakes, everything is fine. The heuristic clustering criterion will not change. But what happens after the classification component has failed several times? That is an indicator that the structure of the overall criterion may not be the most appropriate one for the current external/internal environment. Shouldn't the classification component of a cognitive system give a chance to the elementary criteria that used to have the strength below the threshold? The answer is yes, and the mechanism that makes it happen is rather simple. The pre-set threshold is not really "pre set". It changes its value dynamically. It is defined as a specific percentage of the strength of the currently *best* elementary criterion (the one with the highest strength). As long as the classification component works successfully, nothing will change (except that the threshold may get an even higher value). But, after several failures, the strength of the elementary criteria, forming the heuristic clustering-evaluation criterion, may drop significantly, thus causing the value of the threshold to drop as well. Now, some of the criteria with the low strength

may get activated. If they prove themselves in the new situation, their strength will be increased, thus making them more robust and less sensitive to occasional failures. As a result, the heuristic clustering-evaluation criterion, with its dynamic structure, helps the classification component to maintain its adaptability and flexibility, the characteristics so important for success of any cognitive system.

2.6. The Virtual Model

It is the immediate environment that essentially helps the system to focus its attention on the (attribute, value) pairs at the appropriate level of abstraction for the task at hand to come up with a result of classification that is both useful and meaningful to the end user, whether it was a human, an internal problem-solving process, or something else. The environment supplies the goal and current context of classification. It supplies the set of input events to be classified as well.

The *mental* model has some of its resources activated by the information provided from the environment. A portion of GDN, activated by the goal and context of classification, is one example. A description of the structure of attributes domains, along with the set of inference rules helping the system to climb a domain hierarchy to the needed level of abstraction, is another one.

Now, the *virtual* model can be defined as a collection of declarative and procedural information extracted from the environment and supplied by the mental model, brought together to help the system to solve the task at hand. That is true for any task posed to a cognitive system, and certainly true for the classification task in particular. The virtual model helps the classification component generate useful and meaningful results of classification most of the time. Why most of the time? Why not every time? The reason is that it deals with incomplete information from the environment as well as the incomplete knowledge of the structure of attribute domains. Also, the goal may not be stated clearly. There may be more than one goal. What are the relationships

between them? Is there a hierarchy of subgoals? Are they all independent, instead? These questions and problems just point out the heuristic nature of the whole process, and a need for the heuristic approach to its solution.

2.7. Visibility of Information

One of the most important problems in classification is generation of the list of goal-relevant (attribute, value) pairs, for it is this list that will have a particular influence on the quality of the final result. As pointed out earlier, the (attribute, value) pairs are posted to the list by the rules from GDN that relate goals of classification to an expected context. Also, the (attribute, value) pairs can be posted as a result of the knowledge of the associative links between events and/or concepts. According to the scenario outlined earlier rather vaguely, different types of information will be used at different times during the classification process. The question is, then, how does the system know when to use which information? The parameter called *visibility* of information gives the system an answer to that question.

Visibility of information is determined according to the following criteria:

- relevancy to the goal of classification,
- closeness to the current context of classification.

The more relevant the information to the goal of classification, the more visible it gets. Similarly, the closer to the description of the context the more visible the information. According to the visibility parameter, the rule that matches the goal of information as well as the current context will have a high value on the visibility parameter. The rule that matches the goal but not the context of classification completely, is considered to contain a (attribute, value) pair that is relevant to the goal in a broader sense (common context), and as such should not have as high a score on the visibility parameter as the first rule. The rule that

does not match either the goal of classification or its current context, should have a much lower score than any of the two rules mentioned above.

It seems logical, now, that the (attribute, value) pairs posted to the list of goal-relevant attributes by the GDN rules that match goals of classification as well as all specifics of the current context, should be taken into account prior to any other (attribute, value) pair. The next (attribute, value) pairs to be considered are those posted by the GDN rules that match goals of classification but only some aspects of the current context. The last (attribute, value) pairs to be taken into account by the classification algorithm, in the clustering phase, are those generated after evaluating the associative links of the events to be classified.

Let us notice, at the end, that the strength of a GDN rule lies in direct relationship with the visibility of the (attribute, value) pair to be posted by that rule.

2.8. Classification as a Heuristic Process

The heuristic nature of classification has been pointed out in the analysis of different aspects of classification as a cognitive activity. It may be of benefit to this discussion to restate explicitly what is heuristic about classification at all:

- Classification is an inductive process. Its goal is to form new concepts, given a set of events/instances. There has been no successful attempt at formalizing inductive reasoning. To choose between syntactically valid hypotheses, the system needs some background, both domain-free and domain-specific, knowledge. How does it choose the *right* hypothesis? The answer is task-specific and heuristic in nature.
- Climbing a domain hierarchy, in order to suggest a meaningful and useful clustering at the appropriate level of abstraction, is a heuristic process, driven by the goal and current context of the classification task.

- The clustering phase of the algorithm performs several iterations over the set of given events, when needed, each time with different content in the list of goal-relevant (attribute, value) pairs. The order of iterations is defined heuristically, according to the visibility of information criterion.
- One of the clustering-evaluation criteria is the following heuristic: *a distribution of given events over the newly-generated classes should be roughly uniform.*

2.9. The Problem-Solving Process as a Performance System

Michalski and Stepp [Michalski and Stepp, 1983] [Stepp, 1984] [Stepp and Michalski, 1986] have defined LEF as a criterion that evaluates the quality of the generated classification. Section 2.5.4. of this study explains the reasons for implementing the heuristic clustering-evaluation criterion, instead. But, the system does have to have some way of measuring the quality of the overall result of classification, to be able to use that evaluation as a source of dynamic changes, thus improving its adaptive power.

However, no criterion, no matter how defined, can do the job. The reason is that the evaluation criterion can and will work only in a specific subset of situations interesting to the system. It cannot account for all possible variations so characteristic of all inductive processes. But, what can the system do in that situation? The only way out is to let the end user (the one who initiated the process of classification in the first place) evaluate the result of classification. When the classification is performed by humans, it is the internal and/or external environment, including interactions with other human beings, that is responsible for the final word on the quality of classification. In the case of a machine simulating human cognitive behavior, it is the *problem-solving* process and/or the

external user that will evaluate the resultant classification. The point is that the only acceptable classification-evaluation criterion is a performance system, which will use the result of classification in the intended environment for the intended purpose, to verify its validity and usefulness. It is then and only then that the classification, as an inductive process, will be tested and evaluated in an unambiguous and appropriate manner.

As a result, two kinds of feedback, *internal* and *external*, are identified. Internal feedback is provided by the problem-solving process, which evaluates the classification *externally* to the classification component but still within the model of a cognitive system, as opposed to an internal evaluation of the result of classification, such as using some classification-evaluation criterion. External feedback, on the other hand, is provided by the user external to the system as a whole. The comments, provided by the user, can be of crucial importance to the future success of the system. This situation can be explained the best by the child looking around him/her in search of clues, as additional sources of information, that can help in solving the mystery. The feedback, then, can be used in the process of updating the strength of GDN rules, associative links, clustering-evaluation criterion, different thresholds defined in the system, etc.

It is interesting to notice that, so far, two *levels* of dynamic modifications of the knowledge possessed by the system, the knowledge related to the classification component, have been defined. The first level is the update initiated by the internal evaluation of the result of clustering; while the second one is the update initiated by the external evaluation of the final result of classification. An interesting question is how does the update introduced at the second level modify the update already performed at the first level, within the same classification task? This problem will be discussed in one of the later chapters describing details of the classification algorithm.

At this point, it is clear that the whole process is heavily dependent on the amount and quality of the existing knowledge, and that its performance should improve over time with acquired experience.

CHAPTER 3

A MODEL OF COGNITIVE SYSTEMS DEFINED FROM A PERSPECTIVE OF THE CLASSIFICATION PROCESS

3.1. Classification as a Bottleneck

Holland and his colleagues have devoted much of their book on *induction* [Holland *et al.*, 1986] to a description of their conception of *mental models*. They have defined a mental model as the cognitive system's representation of some portion of the environment. The set of *states* of the environment at time t is denoted by $S(t)$. A *transition* function T describes a change (a transition to another set of states) in the environment. It is represented by the following relationship:

$$T[S(t)] = S(t+1).$$

As the authors have pointed out, given the complexity of environments and the limitations of cognitive systems, it is unreasonable to expect mental models to be *isomorphisms* in which each unique state of the world maps onto a unique state in the model. To reduce the complexity of the environment, the cognitive systems attempt to aggregate environmental states into categories and ignore details irrelevant to the goals of the model. As a result, the mapping from elements of the world to elements of the mental model is *many-to-one*, and is called a *homomorphism*. Consequently, a *categorization function* P , which maps sets of world states into a smaller number of model states, was defined. This was an appropriate point for the authors to introduce a *model transition* function T ,

intended to mimic the *world transition* function T . Given a set of states S' of the model, the following relationship holds:

$$T[S'(t)] = S'(t+1).$$

The function T describes “the manner in which *categories* of environmental states, coupled with *categories* of actions, lead to *categories* of subsequent states” [Holland *et al.*, 1986]. The resulting homomorphic model, along with the relationships described above, is illustrated in figure 3.1 (originally published by Holland *et al.*, [1986, page 33]).

A valid description of the environment, which constitutes a homomorphism, implies *commutativity*: carrying out a transition in the external world and then determining the *equivalence* class of the resulting state has the same effect as determining the equivalence class of the initial world state and then carrying out the transition in the model, i.e.

$$P\{T(S(t))\} = T\{P(S(t))\}.$$

It is obvious that the quality of the mental model is dependent on the quality of a categorization function P . The better the categorization function, the better the description of the outside world; and, consequently, the better the description of the outside world, the more appropriate the actions by the cognitive system as a response to changes in the outside world. Finally, the better the response by the cognitive system, the higher its chances to succeed (whatever the goal).

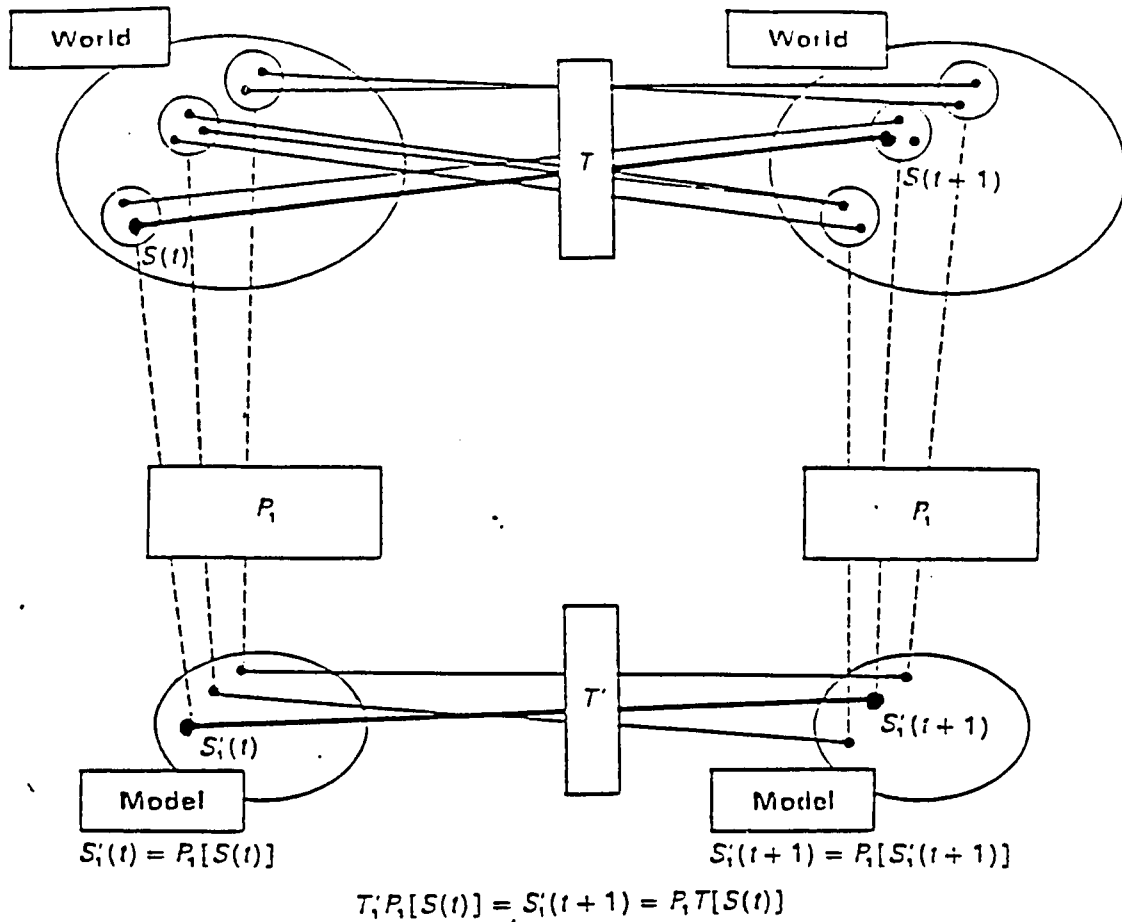


Figure 3.1 Homomorphism.

Unlike Holland and his colleagues, who paid attention mostly to the process of induction in general and analyzed the concept of mental model in that light, this work devotes special attention to the classification process alone, because of its obvious impact on the whole notion of mental model. For that matter, the role and place of the classification process within the cognitive system itself as well as its relationships with the rest of the system had to be carefully analyzed. It was obvious what kind of information the classification process provided to the mental model. However, it was not certain how extensive the information flow was in the other direction, i.e. from the mental model to the classification

component of a cognitive system. The real question was, with what kinds of information did the mental model supply the classification process? This direction has not been adequately exploited either by researchers in the area of conceptual clustering (machine learning) or in the area of categorization (psychology). It was reasonable to assume that not only the classification process defines the quality of the mental model, but the model *guides* the process of classification in describing the outside world as well, thus making the whole process increasingly efficient. The next section will define a model of cognitive systems in greater detail, thus explaining which processes are responsible for what kind of support for the classification component.

But, before turning the attention to the model of cognitive systems, it is important to point out that not all of the processes described in the model have been granted the status of a cognitive process in the psychology literature. An example is certainly the spreading-activation process, which is clearly a subcognitive process. It has been described as such in Chapter 1. However, because of its impact on the classification process, it has been defined explicitly and granted an equal status with all other processes.

3.2. Description of the Model

The model of cognitive systems, first published by Hadžikadić and Yun [1987] in this form, is presented through the sequence of actions performed by the model upon receiving a description of attended events from the outside world. The sequence intentionally emphasizes the role of the classification component, and places the component in its natural environment of interactions with the rest of the system.

A description of a portion of the outside world is captured by the sensory registers (SR). There is a substantial amount of interest devoted to the subject of sensory registers in research done in psychology. It is known that the information

stored in SR remains there for a very short period of time, in much the same form as it was initially presented, until it can be put into a new form and sent further on into the system. The length of time that information is held in SR is brief under any circumstances, because the content of SR is subject to a process of very rapid decay. Also, information can be removed from SR because new information comes in.

The *pattern-recognition* process is depicted as an intervening process between the sensory registers and working memory (WM). Pattern recognition is the process of matching incoming sensory information with previously learned information stored in the knowledge base (KB), thus effectively converting raw information to data meaningful to the system.

However, the pattern-recognition process is *syntactic* in nature. It does not assign a meaning to the recognized object. It should recognize, for example, a letter *A* as a first letter of the alphabet, no matter how poorly written. But, the pattern-recognition process is not responsible for telling the system what meaning that letter has in the current context, whatever it happens to be. Since the approach described in the study deals with induction in the context of classification, which is anything but syntactic in nature, the pattern-recognition process is skipped and replaced with the classification component, thus emphasizing two aspects of a cognitive system in general:

- 1) The system is interested in a description of the environment not in its syntactic form, but rather in the form of concepts and their relationships that adequately represent the environment with respect to the current goals and state of the cognitive system, thus introducing the possibility of viewing the same events differently if appropriate in the current situation;

- 2) The system is interested in an ability to learn new concepts after seeing events it doesn't recognize as instances of any concept known to the system already. Thus concept recognition and concept formation are the reasons for the important role the classification process plays within the cognitive system.

The relationship between the classification process and *architectural* components of the model of a cognitive system is depicted in figure 3.2. A *square* represents an architectural component, while an *ellipsis* stands for a *cognitive* process. Notice that the figure describes the concept recognition role of the classification process. It stores a *meaningful* description of the environment in the working memory. Later on, the classification process will be invoked again, once the problem-solving process needs its services. It is then when the classification process performs the role of concept formation. Then figure 3.2 will be referred to once again.

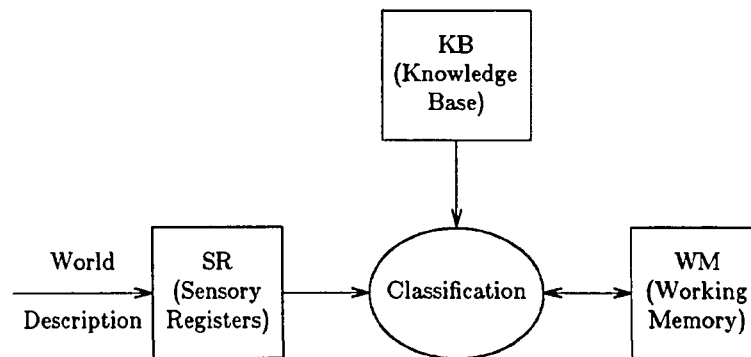


Figure 3.2 Environment of the classification process.

However, at this point we know nothing about the internal structure of the working memory and knowledge base. So, their description is in order before proceeding any further. The working memory resembles the properties of what is known in psychology literature as short term memory (STM). It is of limited

capacity and consists of four components (figure 3.3):

- agenda,
- active rule set (ARS),
- virtual model state description (VMSD),
- external world state description.

Agenda	ARS (Active Rule Set)	VMSD (Virtual Memory State Description)	EWSD (External World State Description)
--------	--------------------------------	---	---

Figure 3.3 Working memory (WM).

The agenda is a list of goals posted by activated processes of the system, to be fulfilled by some other processes. If there are no goals on the agenda, a *default* goal of initiating a constant interaction with the environment is activated. The list of goals will initiate a process of retrieving a set of rules capable of satisfying posted goals in the current state of the external world and its internal description. Those rules are kept in the active rule set (ARS).

VMSD contains a description of the current state of the virtual model. Similarly to a *virtual copy* of Fahlman [1979] and a *mental model* of Holland and his colleagues [Holland *et al.*, 1986], a *virtual model* includes a description of the immediate environment, along with the resources (procedural and declarative knowledge) that the mental model has activated to have incoming information processed appropriately. In other words, VMSD contains information describing what the cognitive system “believes” exists outside (surrounding it). What really is outside or, to be exact, what properties of the outside world the cognitive system has attended to, is stored in the EWSD portion of the working memory. At

this moment, the virtual model can be defined as the content of WM from the moment of initiation of the problem to be solved to the moment of its actual solution.

The knowledge base (KB) partially resembles the long term memory, and contains declarative as well as procedural knowledge of different types (some types of knowledge, listed here, are more completely described by Holland *et al.* [1986]): control knowledge, diachronic predictor and effector rules, categorical and associative synchronic knowledge, inferential rules and heuristics, evaluation-control rules, spreading-activation rules, context extraction and generation rules, analogical-mapping rules, encoding rules, matching and retrieval rules, etc. Each of them is described briefly.

Control knowledge comprises information about the default goals as well as the rules responsible for detecting the difference between the states of the virtual model and external world.

Diachronic rules specify the manner in which the environment is expected to change over time, either autonomously or in response to outward-directed actions of the system. Diachronic rules are divided into two classes: *predictor* rules, which tell the system what to expect in the future, and *effector* rules, which cause the system to act on the environment.

Synchronic (procedural + declarative) knowledge is an *atemporal* type of knowledge. It provides recategorizations of, and associations with, events and concepts at a single time. Synchronic knowledge represents the kind of information typically represented in a semantic net. It is further subdivided into *categorical* and *associative* knowledge. Hierarchical category relations are primarily given by categorical type of synchronic knowledge. It provides the basis for determining category membership, for reclassifying concepts, and for assigning properties to them. Associative knowledge relates concepts that have nonhierarchical

relations, thus allowing one concept to remind the system of another concept by activating it in memory (KB). The function of diachronic rules and synchronic knowledge, taken together, is to model the world.

The primary function of *inferential* rules and heuristics is to produce better diachronic and synchronic rules efficiently. They are abstract enough to be applicable to a broad range of domains.

Evaluation-control rules help the system to evaluate the strength of candidate rules for the task at hand and select the ones that score above the predefined threshold.

Spreading-activation rules tell the system how to make use of synchronic knowledge in its attempt to activate the descriptions of the events associatively related to the events already encountered in the current context.

Context extraction and generation rules are responsible for detecting regular co-occurrence of properties of an environment and explicating them in a form useful to the system.

Analogical-mapping rules guide the system in locating a plausible source of analogy and aspects of the source relevant to the solution of the *target* problem.

Encoding rules are extremely important for the flexibility of the system, because of the fact that an appropriate representation (suitable for the task at hand) will significantly contribute to the efficiency, effectiveness, and ease of execution of any cognitive process.

Matching and retrieval rules are responsible for the efficient retrieval process. It is obvious that these rules are closely related to the encoding rules, for the system must know how the specific information has been stored in order to retrieve it when needed.

In figure 3.4, KB is represented as a square (an architectural component of

the model of cognitive systems) with the constituting types of knowledge listed and numbered. These numbers will help keeping the final figure of the model, to be given later in the section, simple and readable.

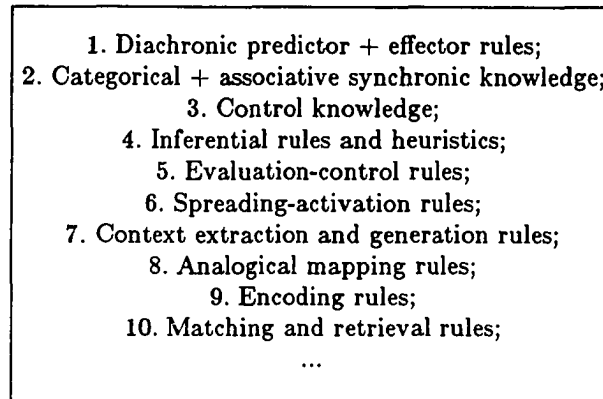


Figure 3.4 Knowledge base (KB).

Once a description of the outside world has been stored in the EWSD portion of WM, the control process takes over and compares the contents of EWSD and VMSD, i.e. the current states of the environment and the mental model respectively. As a result, it establishes the goal of *reducing the differences*. During that process, the control process relies on the control knowledge from KB. If there are no differences between the two states, it maintains a default goal on the agenda. Figure 3.5 depicts the environment of the control process within the model of cognitive systems.

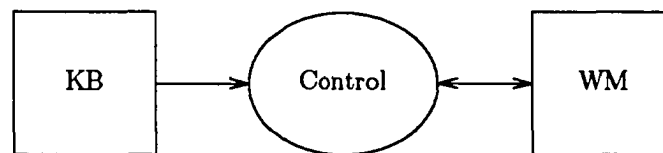


Figure 3.5 Environment of the control process.

It is the problem-solving process that is supposed to take the goal with the highest priority on the agenda and try to come up with a solution to the problem

at hand. It consists of two processes: (a) the *evaluation* and (b) *state transition* process. The evaluation process, based on the content of ARS, rules from KB (evaluation-control rules, inferential rules and heuristics, and analogical-mapping rules), and a content of the episodic memory (EM), evaluates the strength of each candidate rule to fire and selects one with the highest score. The actual process of firing a rule is embodied in the state transition process, which makes use of diachronic predictor and effector rules to translate the virtual model from one state to another, thus effectively emulating changes in the environment. The relationship between the components of the problem-solving process and components of the model of cognitive systems is given in figure 3.6.

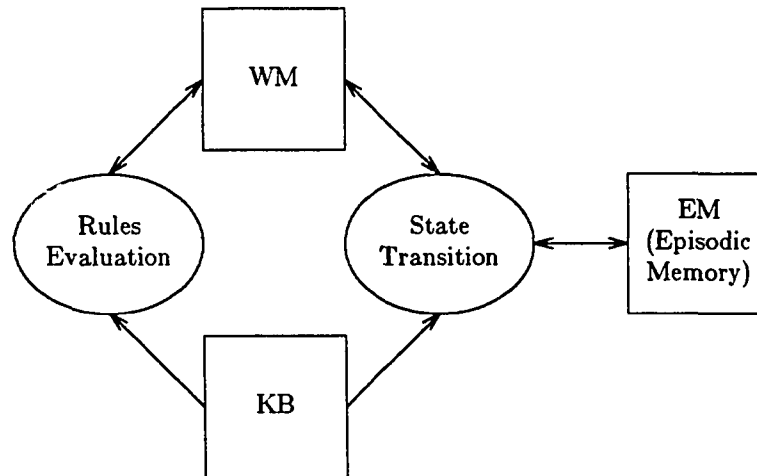


Figure 3.6 Environment of the problem-solving process.

There is a piece of this figure not previously explained. Episodic memory (EM) keeps track of episodes from the system's experience and constant interactions with the environment. Based on Schank and Abelson's notation [Schank and Abelson, 1977], these episodes will be called *scripts*. Scripts help a cognitive system to generate a set of expectations, when confronted with few aspects of a known situation.

At this point, one more definition of a *mental model* can be introduced: a mental model is defined by the overall information contained in both KB and EM, thus representing the experience of the system gathered through interactions with the outside world.

Also, the *long term memory* (a term used in psychology for the storage of our knowledge of the world) can be defined as a combination of two components from the model of cognitive systems, specifically KB and EM.

While EM keeps track of episodes in the form of scripts, there must be a process that generates them. It is the *context-features extraction* (i.e. scripts creation and update) process that consults the context extraction and generation rules as well as the description of the environment in the EWSD portion of WM, in order to create or update a script that captures aspects of the environment relevant to the cognitive system. This situation is explicated in figure 3.7.

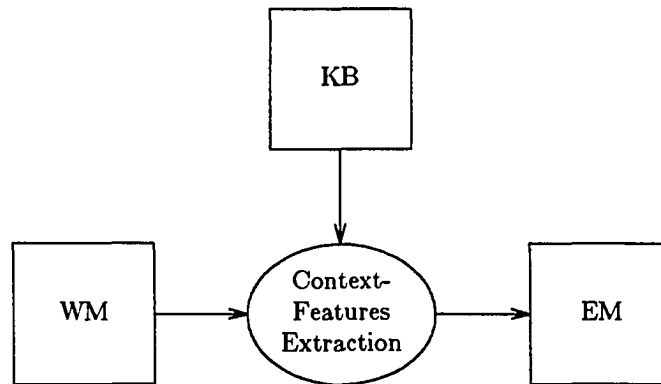


Figure 3.7 Environment of the context-features extraction process.

The *expectations-generation* process is supposed to pull out the most appropriate script from the “library” of scripts, generated by the context-features extraction process, once requested by the problem-solving process in the course of finding a solution for the task at hand. In order to do that, the expectations-generation process needs the information from EM, appropriate

rules from KB, and a description of the problem environment from WM (figure 3.8).

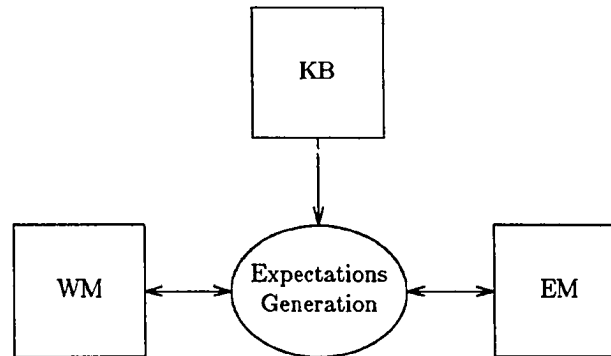


Figure 3.8 Environment of the expectations-generation process.

Very often a cognitive system finds it useful to explore associative links between events and/or concepts. Whenever there is no plausible solution to the problem by exploiting hierarchical links, i.e. *superordinate-concept*, *subordinate-concept*, *instance-of*, *has-instances*, it may pay-off to turn attention to non-hierarchical links and utilize information of a different nature, much like an analogy to a similar problem already solved can help us solve another problem in a different domain. The *spreading-activation* process, based on the content of VMDS and/or EWSD and spreading-activation rules from KB, will activate the events/concepts associatively related to the events/concepts currently attended by the system (figure 3.9). This *advancement* along the associative links happens one step at a time, for the reason explained earlier in Chapter 2.

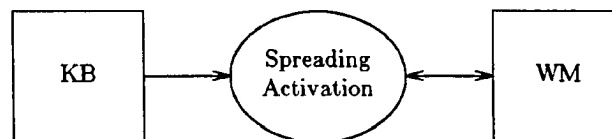


Figure 3.9 Environment of the spreading-activation process.

Once the system is confident that the knowledge gathered through the interaction with the outside world (including the results of the problem-solving process tested in the environment) is worth saving for possible future use, the *encoding* process is initiated to actually perform the storing process. It will store the content of ARS, VMDS, and/or EWSD in KB (figure 3.10). This process is responsible for the update of rule and heuristic strengths, (attribute, value) pair relevances, and threshold values as well. It is not to say that this process will compute those updates, but rather that it will perform the physical storage of new values already calculated by some other cognitive activity. The important function of the encoding process is a *garbage collection*. After saving the relevant content of WM, it may clean up WM and prepare it for the next information about the external world of relevance to the cognitive system.

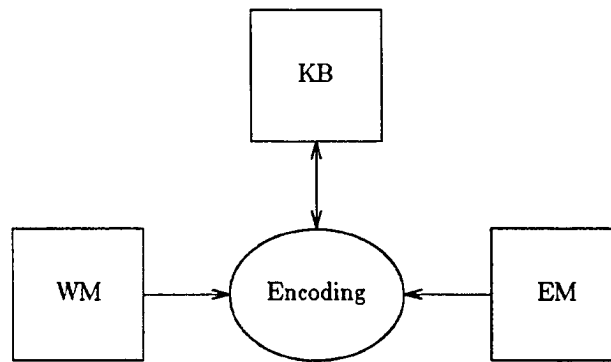


Figure 3.10 Environment of the encoding process.

An opposite process to encoding is *matching and retrieval*. The matching and retrieval process matches the content of VMDS, EWSD, and/or ARS with a content of KB and retrieves the knowledge structures that have obtained a sufficient amount of *support* (figure 3.11). Since the cognitive system has to know the storing mechanism when retrieving information, the matching and retrieval process relies on encoding rules as well as on matching and retrieval rules and categorical

+ associative synchronic knowledge.

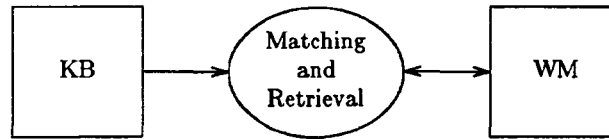


Figure 3.11 Environment of the matching and retrieval process.

Once all the components of the model of cognitive systems are defined, the whole system can be presented at once. Figure 3.12 represents the model of cognitive systems viewed from the perspective of the classification process. Notice that each link from KB to a specific process has been indicated by the label “KB”, followed by numbers included in parentheses, rather than actually drawn. The numbers stand for the corresponding types of knowledge from KB used or worked on by that process. If there is no number in the label of a link, it means that the whole content of KB is reachable by the process in question. This labeling makes the figure easier to analyze and understand.

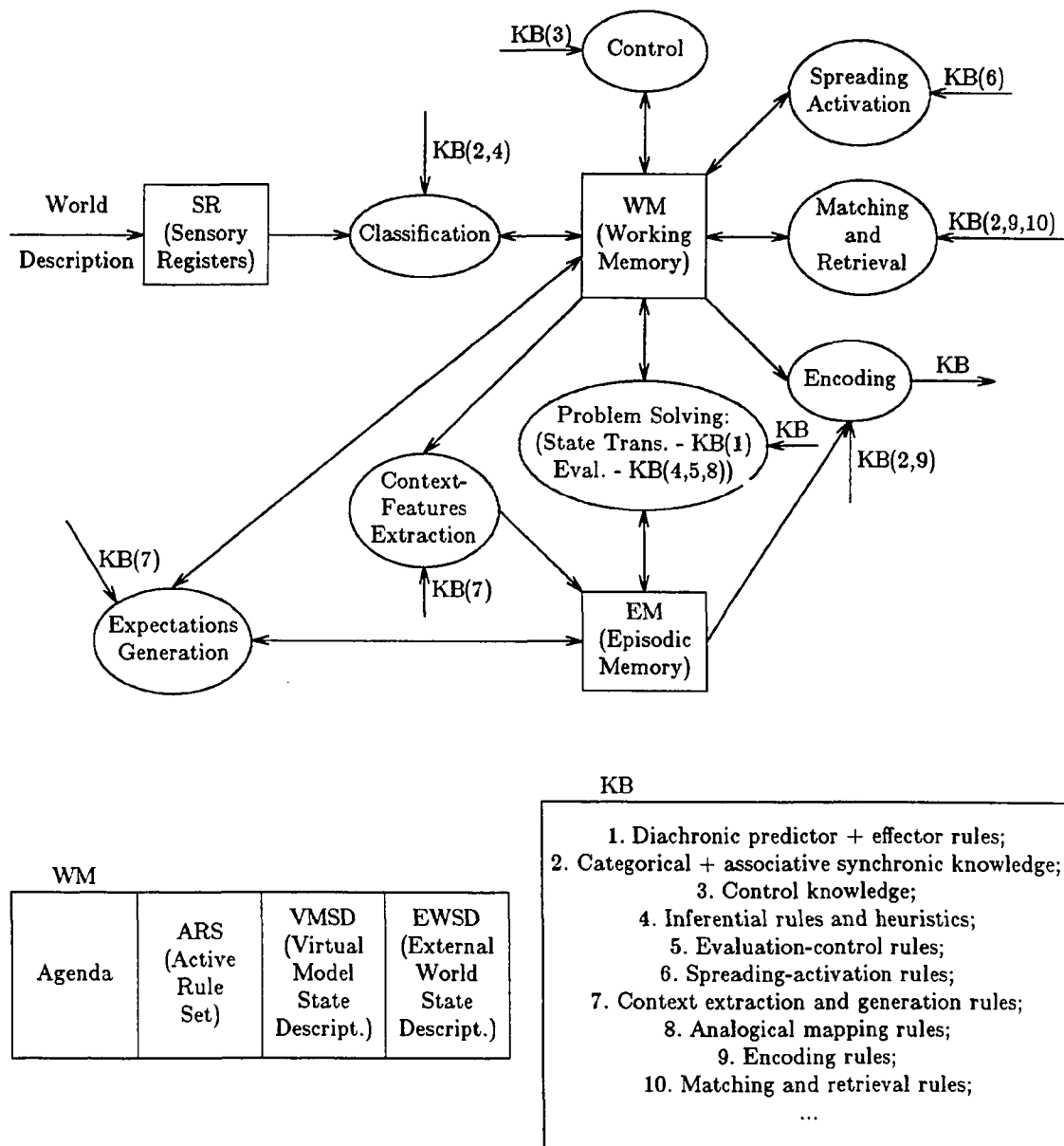


Figure 3.12 A model of cognitive systems viewed from the perspective of the classification process.

Earlier in this section the *concept recognition* aspect of the classification process has been described. At this point, since the model of cognitive systems has been defined in greater detail, it is time to describe the *concept formation* aspect of the same process. Let's assume that the problem-solving process, after taking over control in order to satisfy the goal with highest relevance from the agenda, discovered that it needs help from the classification process. It posts an appropriate goal to the agenda for the classification process and provides in VMSD the set of events/concepts to be classified along with a description of the current state of the virtual model. Since the problem-solving process cannot resume its course of action without needed data, the classification process steps forward. It attempts to classify given events based on information it has, i.e. the goal and context of classification (WM), categorical + associate synchronic knowledge, and inferential rules and heuristics (both from KB, both emulate GDN) as well as information ready to be supplied, if needed, by the expectations-generation and spreading-activation processes. What kind of information will be used in the process of concept formation and in which order, is defined by the *visibility of information* parameter previously defined.

Once the classification is generated, the problem-solving process gets activated again. It will try to solve the problem at hand by using the result of classification on the solution path. The attempt may or may not be successful, but whatever happens, the problem-solving process must inform the classification process about the outcome. Based on this internal feedback, the classification component of the cognitive system will react appropriately and increase/decrease the amount of its *belief* to the rules, heuristics, and thresholds that have made a contribution to the generated classification. This process may go through several iterations before the final solution to the problem at hand has been found. Once the solution has been reached, the encoding process may take over, store the

result of classification, save the updated values, and do necessary garbage collection.

The important aspect of this discussion is the role of the problem-solving process. With its feedback to the classification process, the problem-solving process effectively emulates a *performance system*, which is of crucial importance to any learning system. No criterion of the quality of classification can substitute for the role of the end user whoever/whatever that may be. It would be virtually impossible to produce a psychologically plausible classification without ability to verify the result of classification in its original environment. It is the feedback from the end user that keeps the system learning from each failure as well as success. It is the feedback that makes this inductive process manageable and unambiguous.

3.3. Implications of the Model

What are the implications of the model of cognitive systems? There are several of them. The model defines:

- 1) which processes are responsible for supplying what information;
- 2) which components of the model and what type of information constitute a mental model;
- 3) which components of the model and what type of information constitute a virtual model;
- 4) importance of the parameter of visibility of information;
- 5) importance of the problem-solving process as a performance system for the classification process;
- 6) flexibility of the *process-synchronization* mechanism based on the *post_a_goal - wait_for_result* and *posted_goal - generate_a_result* kind of protocol;

- 7) existence of two levels of parallelism: (a) *intra-process* and (b) *inter-process* parallelism.
 - a) The context-features extraction process can be executed parallel with, for example, the classification process. The former process does not change the data used by the latter process to produce a classification. It only recognizes regularities in their structure.
 - b) There is a substantial amount of parallelism within each process itself. For example, during the evaluation phase of the problem-solving process all candidate rules could be evaluated in parallel.

After defining the position of the classification component within a cognitive system, it seems to be appropriate to focus attention on the internal organization of the component itself. However, before doing that, the representational issues are going to be discussed first, for the classification component must be aware of the manner in which the information it works upon has been (or should be) stored in the memory.

CHAPTER 4

KNOWLEDGE REPRESENTATION

4.1. Representation and the Cognitive Approach

There has been a substantial interest in cognitive psychology for *mental* representation of the *world*, *perceptual* in particular. Many theories and representational mechanisms have been developed and suggested. They include templates, features, structural descriptions, first-order and second-order isomorphism, and prototypes. Although each of them will be described briefly, a thorough analysis can be found in Palmer's article [1978].

4.1.1. Templates

A *template* is very often defined as a figure that displays a digitized pattern overlapping, to a certain extent, with an input pattern. The simplest case is a *standard* template which is matched against the input pattern without any preprocessing. The match is carried out in a point-to-point fashion. If there are n positions, the number of matches can vary from no points to all n of them. Patterns are then classified according to some decision strategy, usually the *best-fit*. Since trivial changes in position, orientation, and/or size could have catastrophic consequences for classification performance, preprocessing operations have been suggested to normalize (translate, rotate, clean-up, etc.) the input-pattern prior to matching. This operational aspect is a characteristic of what is called *preprocessed* templates. *Hierarchical* templates are characterized by the fact that the components of a complex pattern might be a set of simpler templates rather than just a set of points. These simpler templates could then be defined by even simpler templates and so on, until the individual points at the terminals have

been reached. The method itself is more powerful in its ability to represent the world, with an obvious drawback in increased processing complexity of the decision process.

In general, templates is a fairly well-defined theory that has recaptured the interest of researchers after the advancements in the theory of prototypes and work on image rotation. Also, templates seem to be an appropriate mechanism for representation of low-level visual information.

4.1.2. Features

Feature representations were meant to be an alternative to templates. The main reason for their popularity lies in the flexibility of the mechanism, *anything can be a feature*. This fact is the source of their greatest strength and greatest weakness. As Palmer pointed out [1978, page 282], "it makes them convenient to use to explain data, but it makes them inherently ill-defined as a theory." The most popular feature theories are binary features, multidimensional spaces, and hierarchical features. *Binary* features theories operate as follows:

- a) A set of n operational feature tests are applied to an input pattern. There are two possible outcomes of each test, *yes* or *no*, with respect to the presence of the feature.
- b) The results of the tests are compared to a set of stored representations of pattern types, each one defined as a list of values for the same features (indicating their presence or absence). For each feature, a match is recorded if both the input pattern and stored representation have the same value, either *yes* or *no*.
- c) Some strategy of computing the measure of similarity is employed, usually the total number of matches. Various weighting coefficients can be introduced to reflect the saliency of different features.

- d) The resulting measure of similarity is then used to classify the pattern according to some decision strategy.

Note that if the features are exclusively *position - (black or white) color*, the result is a standard template theory. Thus standard templates appear to be a special case of binary features.

Multidimensional space representation, on the other hand, describes an event as a point in n -dimensional space. Relations among groups of events are preserved by spatial relationships among sets of points. In order to classify the input pattern, again a set of n operational feature tests must be performed, but this time each test has m possible outcomes rather than only two, where m may be an infinitely large number. The outcomes represent the *degree* to which the event has a specific feature. The results of the tests specify the point in the n -dimensional space occupied by the pattern. The point is then compared to a set of predefined representations of pattern types. There are two possible decision strategies: the *point* method and the *region* method. Which one is chosen depends on the representation of stored patterns. If the stored representations of pattern types are single points in metric space, some form of distance metric is used, usually Euclidean. The pattern then belongs to the category at the shortest distance from the pattern, thus effectively emulating the point method of the decision process. In the region method, stored patterns (classes) are represented as a region in space. The input pattern is classified as an instance of the class within whose region it falls.

The basic idea behind the *hierarchical* features approach is that complex features can be defined in terms of more *primitive* ones; thus, feature dimensions are structured according to their relationships. Hierarchical feature theories are one of the possible ways to specify logical dependencies that exist among different dimensions. Since the structural relationships among different dimensions can be

represented, the mechanism of hierarchical features can account for the fact that people are able to analyze several dimensions combined into a unitary aspect of the stimulus much more easily than any component dimension separately.

4.1.3. Structural Descriptions

Some researchers have rejected both features and templates as representations for pattern recognition because of their shortcomings in describing the structural relationships between patterns and their parts. The *structural descriptions* theories, whose power of representing relations on more than one event differentiates them from features, have been suggested. The basic idea behind *simple* structural descriptions is that a pattern is defined by relationships among subpatterns. A subpattern, in turn, can be either a primitive or relationships among further subpatterns. The difference between simple structural descriptions and hierarchical templates lies in the fact that the relations among subpatterns can vary.

A potential drawback of simple structural descriptions is that their representations still rely on just positions and *colors* and all properties must be derived from them. What happens when parts of a higher order, formed by relationships among component parts, have properties not defined for the components? A possible solution to this problem is to augment simple structural descriptions with features for the patterns of higher order. The result is *augmented* structural descriptions, with a general assumption that any pattern is represented “*both* as a set of unary dimensional values and as a set of relationships among component parts” [Palmer, 1978, page 287].

4.1.4. First-Order and Second-Order Isomorphism

In the concept of *first-order isomorphism* the properties of events from the world are retained in the internal representation of those events. In other words,

the representation of a green square must be itself both green and square. An alternative to first-order isomorphism is *second-order isomorphism*, where the internal representation of a square need not be itself square, but it must be functionally more similar to a rectangle than to some unrelated event. The emphasis is on *functional* sameness rather than on *physical* sameness. This functional correspondence is what essentially decouples the external and internal worlds in terms of resemblances.

4.1.5. Prototypes

Prototypes are highly specific categorical representations stored in memory. They approximate the *most typical* or the *ideal* instance of the category. The input pattern is represented along the same dimensions as the prototypes. A measure of similarity, continuous in nature, is computed between the input pattern and *each* categorical prototype. Based on the degree of similarity, the pattern is assigned to a category according to some decision strategy. The most common classification rule is the *best-fit*, resulting in the pattern being classified into *one-and-only-one* category.

The prototype approach is considered to be in opposition to the *invariant attribute* approach. Invariant-attribute representations of categories/classes cannot represent dimensions that vary within categories, but only dimensions that vary across categories. The main assumption of those theories is that the representation of categories should be very general, such that each instance is completely and equally consistent with it. An example of the invariant attribute approach is the *Annotated Predicate Calculus* language, to be described in the next section. However, a prototype representation has relatively high resolution for dimensions of information that vary within the category. That was the reason behind the statement at the beginning of this section that prototypes are highly specific categorical representations. This *specificity* is with respect to

within-category variation [Palmer, 1978]. There is a broad range of possible theories between the two extreme approaches, prototypes and invariant-attributes approaches. They would differ in their representation of within-category variation.

Prototype theories are a general class of theories that do not require any assumption about the nature of dimensions or how they are represented. It is only when people have in mind a specific category that those assumptions are needed. As a result, any theory of perceptual representation will have to be consistent with the notion of prototypes, for it has to be able to represent highly specific instances. As Palmer pointed out [1978, page 290], “prototypes are a construct of categorical representations, not of representations in general. As a class, they are equally compatible with virtually any theory that can represent specific instances.”

4.2. Annotated Predicate Calculus

Many researchers interested in methods of inductive learning have used a restricted form of *predicate calculus* as a representational mechanism in their work (Vere [1975], Michalski [1980], etc.). Some other formalisms include *decision trees* [Quinlan, 1983], *production rules* [Waterman, 1970], *semantic nets* [Haas and Hendrix, 1983], and *frames* [Lenat, 1983]. The *Annotated Predicate Calculus (APC)*, a language created by Michalski [1983], is an extension of predicate calculus that uses several novel forms and attaches an *annotation* to each predicate, variable, and function. The annotation is a store of information about the given predicate or atomic function, for example the definition of the function's value set. Since APC has been used in CLUSTER/2 and CLUSTER/S conceptual clustering algorithms [Michalski and Stepp, 1983] [Stepp and Michalski, 1986], and created with application to clustering problems in mind, it will be described here to a certain extent. Examples used in the discussion are taken

from Stepp and Michalski [1986, pp. 487-488].

APC supports all forms found in predicate calculus. In addition, it employs a special kind of predicate called a *selector* in the following form:

$$[\text{atomic-function REL value-of-atomic-function}]$$

where REL (relation) stands for one of the symbols $= \neq < > \leq \geq$. An example of a selector is

$$[\text{weight}(\text{box}) > 2\text{kg}]$$

with the following meaning: “the weight of the box is greater than 2 kg.” The notation $f(a, b, c, \dots)$ denotes a multi-argument function when the position of arguments is important, otherwise $f(a.b.c. \dots)$ is used. In the case of two-place predicates, $p(a, b)$ denotes an anti-symmetric predicate, while $p(a.b)$ denotes a symmetric one with $p(a.b) = p(b.a)$.

More complex selectors involve *internal disjunction* or *internal conjunction*. These operators apply to terms rather than to predicates. Two corresponding examples are given below:

$$[\text{color}(\text{box}) = \text{red} \vee \text{purple}]$$

$$[\text{color}(\text{box1} \ \& \ \text{box2}) = \text{red}]$$

More complex expressions are obtained by using standard logical operators to combine selectors.

Background knowledge can be described as a set of APC implicative rules:

$$\text{CONDITION} \Rightarrow \text{CONSEQUENCE}$$

where both CONDITION and CONSEQUENCE are conjunctions of selectors. As an example of the implicative statement, the assertion “vegetables are food”

would be expressed as

$$[\text{is\_vegetable}(\text{object1})] \Rightarrow [\text{is\_food}(\text{object1})]$$

If we consider “vegetable” and “food” to be elements of the tree-structured domain of the attribute “type”, an alternative way to express the same assertion would be:

$$[\text{type}(\text{object1}) = \text{vegetable}] \Rightarrow [\text{type}(\text{object1}) = \text{food}]$$

As Stepp pointed out [1984], a conjunctive statement can describe both a single example and a class of examples, depending on the generality of the statement itself. If the universe is the feature space in which all examples are described, then the statement describing a single example would cover only one point in the space of all events (*event space*). A more general statement would cover some part (region) of the event space. Among all points covered by the general statement some points correspond to given examples (*observed events*) and some don't (*unobserved events*). The best fitting statement that covers a given set of observed events with minimal generality is the one that covers the fewest unobserved events.

Further details on APC are given by Michalski [1983].

4.3. The Chosen Representation

The evidence that *prototypes* play a critical role in human categorization is compelling. Rosch has demonstrated [1973] [1978] the existence of prototypes for both natural categories (like colors and animals) and artificial categories (like dot patterns and schematic drawings). Since the main goal of this research effort was to define and implement an algorithm that would produce a meaningful and useful (to people) classification of the set of given events, the representational mechanism adopted in this work was based on the notion of prototypes.

A *schema* mechanism is adopted to implement a concept of prototype. A schema is a *declarative* structure that organizes pieces of knowledge related to the same entity into a unitary whole. In various implementations, schemata correspond to Minsky's *frames* [1975] and Schank and Abelson's *scripts* [1977]. A *slot* is the place where the specific information fits within the larger context created by the schema. Each slot has a name and contains information of specific type. Slots can be operated upon by the *procedural* knowledge implemented in the form of *if-then* rules. Slots can be added or removed, and their content modified.

A template of the schema structure used in this work, representing a concept $schema_i$, is given below:

name:	$schema_i$
function:	$(function_j, relevance_j)$
feature:	$(feature_k, value_k, relevance_k)$
specialization-of:	$(superordinate-concept_l, strength_l)$
generalization-of:	$(subordinate-concept_m, strength_m)$
instance-of:	$(concept_n, strength_n)$
has-instance:	$(instance_q, strength_q)$
associatively-related-to:	$(schema_r, strength_r)$
consists-of:	$(part_t, relevance_t)$

Instead of describing each slot in turn in general terms, the description of a familiar object *chair* in the schema mechanism is given below:

name:	chair
function:	(to-sit-on, 1.0)
feature:	(number-of-legs, 4, 1.0)

	(number-of-legs, 3, 0.25)
	(style-of-back, straight, 0.8)
	(style-of-back, cushioned, 0.5)
	(number-of-arms, 2, 0.5)
	(number-of-arms, 0, 0.5)
	(number-of-arms, 1, 0.2)
specialization-of:	(furniture, 1.0)
generalization-of:	(John's-chair, 1.0)
	(Susan's-chair, 0.8)
has-instance:	(John's-chair, 1.0)
	(Susan's-chair, 0.8)
associatively-related-to:	(being-tired, 0.7)
consists-of:	(seat, 1.0)
	(legs, 0.9)
	(back, 0.7)
	(arms, 0.5)

There are several aspects of this representation that need to be pointed out:

- 1) Slots may have multiple pieces of knowledge of the same type.
- 2) There are two types of links:
 - a) the *specialization-of* and *generalization-of* slots implement a hierarchical tree-like structure with events at the leaves and concepts at the nodes;
 - b) the *associatively-related-to* slot is responsible for building a network-like structure connecting events/concepts.
- 3) An event/concept cannot have both the *instance-of* and *has-instance* slots in its description (cannot be both an event and a concept at the

same time).

- 4) The *function*, *feature*, and *consists-of* slots contain the *relevance* variable. The relevance of the (feature, value)/function/part is inversely proportional to its variability across the descriptions of class members.
- 5) The rest of the slots (the *link*-slots) contain the *strength* variable. The strength of the link is directly proportional to the *similarity* of the schemata. In the case of hierarchical links, the similarity is defined by the degree of match between the function slots and (feature, value) pairs of the two schemata. In the case of the associative links, the similarity is a consequence of the system's experience.
- 6) Since this study describes the classification of unstructured events, the *consists-of* slot is not going to be utilized. It was mentioned at this place for the sake of completeness of the chosen representational mechanism.

An example of a schema description of an event from the chosen application domain, a subset of UNIX user commands, is presented below:

```
(defschema troff
  "text formatting and typesetting"
  (attribute (function text-formatting-and-typesetting 1.0))
  (is-a general-utility)
  (attribute (domain file 1.0))
  (attribute (range formatted-file 1.0))
  (attribute (type-of-parameters file 1.0))
  (attribute (number-of-nonoptional-parameters 0 1.0))
  (attribute (number-of-optional-parameters 100 1.0))
  (attribute (processing-time input-dependent 1.0))
  (attribute (input-device standard-input 1.0))
  (attribute (output-device graphics-systems-phototypesetter 1.0))
  (attribute (meaningful-mnemonic no 1.0))
  (attribute (number-of-flags 14 1.0)))
```

Notice that the relevance of all (attribute, value) pairs is 1.0 in this example. It is because of the fact that any event represents a class with one member, and as

such has all (attribute, value) pairs invariable across the class, which is reflected by the value of relevance of 1.0.

The next example is generated by the system as a result of the *characterization* subprocess, and represents a description of the class with four instances (one of which is the example given previously).

```
(defschema SCHEMA-1
  (concept-attribute (FUNCTION TEXT-FORMATING 1.0))
  (concept-attribute (DOMAIN FILE 1.0))
  (concept-attribute (RANGE FORMATED-FILE 0.75))
  (concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
  (concept-attribute
    (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.75))
  (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0))
  (concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
  (concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
  (concept-attribute (MEANINGFUL-MNEMONIC NO 0.75))
  (instance (NROFF 0.9166666665))
  (instance (STYLE 0.6666666665))
  (instance (TROFF 0.9166666665))
  (instance (TROFF-T 0.9166666665)))
```

Note that the change of the slot name *attribute* in the event description to the slot name *concept-attribute* in the concept description has been caused by limitations of the implementation language (ART). The same is true for the slot *instance* that was supposed to be labeled as *has-instance*.

CHAPTER 5

CLASSIFICATION PROCESS

5.1. Components of the Classification Process

After defining the model of cognitive systems as well as the representational mechanism, the focus of attention can be switched to the classification component (process) itself. As defined previously, the process of classification consists of three subprocesses:

- clustering
- characterization
- building a hierarchy

5.1.1. Relationships among the Components

Several conceptual clustering algorithms and some specifics of their clustering and characterization components have been discussed in Section 1.2.7. As Fisher and Langley [1985] pointed out in their analysis of known conceptual clustering algorithms, the search for clusterings and the search for characterizations are embedded within a higher level search through the space of classification trees (*hierarchies* of concepts). The relationships between the components of conceptual clustering algorithms are shown in figure 5.1.

5.1.2. An Alternative View of the Relationships among the Components

The approach to classification defined in this work varies from those mentioned previously. Based on the assumption of goal-driven classification, and a notion of GDN, virtual model, and mental model, different relationships between

the components of the classification process have been established. But, before the system starts classifying given events, the list of goal-relevant (attribute, value) pairs should be prepared. That includes matching the goal of classification and the current context with the left-hand sides of the rules forming GDN. Once the list is ready, the system has to take into account the structure of attribute domains as well as the descriptions of given events. This process is likely to reduce the size of the list of goal-relevant (attribute, value) pairs and enrich the event descriptions. In order to emphasize the importance of *process preparation*, it has been given the status of a *component* of the classification process. Details of the process itself will be given in the next section.

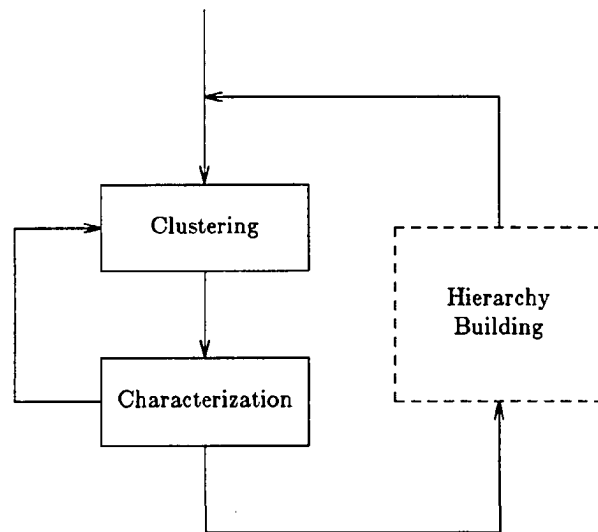


Figure 5.1 Relationships among the components of the classification process.

Once the process preparation is over, the clustering process may proceed with its attempt to produce the number of classes that will satisfy the heuristic clustering-evaluation criterion. According to the discussion from Chapter 2, there is no reason to introduce the characterization process in this loop, if the

representational mechanism has been adequately chosen. This is exactly what is suggested by the approach adopted in this work. The characterization process will be activated once the clustering process is over. Consequently, characterization is just the first step of the process of building a hierarchy (classification tree), which proceeds by invoking the characterization process at different levels of the hierarchy, until it reaches the root node.

A note is appropriate to point out the inaptness of the term *building*. The hierarchy has been built already by the clustering process. The only thing left to be done is to describe the classification tree. As a result, the appropriate name of the process seems to be *describing a hierarchy* rather than *building a hierarchy*. However, to avoid possible confusion when comparing this method to the ones suggested by other authors, the latter term will be used in the rest of the study.

An outline of the defined approach to the classification process is given in figure 5.2.

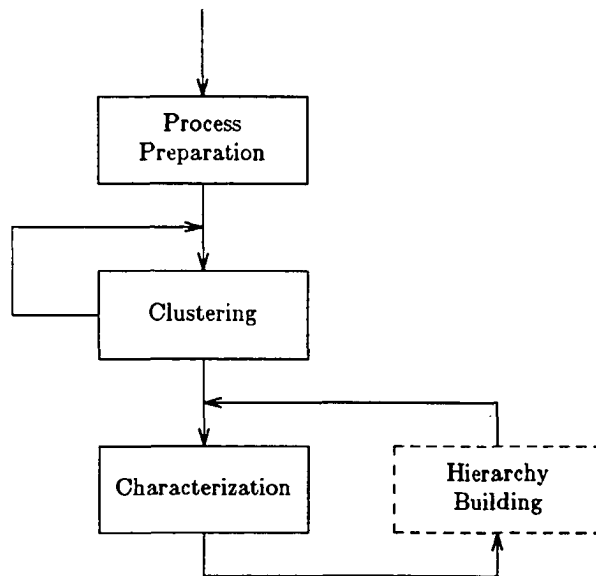


Figure 5.2 An outline of the implemented approach to the classification process.

5.1.3. Comparison of the Views

When comparing the strategies outlined in figures 5.1 and 5.2, it is obvious that the approach defined in this work has the advantage in terms of efficiency in the clustering, characterization, and hierarchy-building parts of the algorithm. For one, characterization is not a part of the loop driven by the search of the clustering component for a plausible clustering of the given events. It will come into play when the situation is safe and the clustering has been decided on. Secondly, the clustering process never evaluates a node that is not on the solution path, if the solution can be found in that phase of clustering. That is the promise of the goal of classification, current context, GDN, and mental model. Notice that three phases of clustering have been defined earlier: (1) clustering driven by the goal of classification and the current context; (2) clustering driven by the goal of classification and the common context; and (3) clustering driven by the associative relationships of the given events. Finally, the hierarchy-building process performs a degenerate search only, while describing the hierarchy generated by the clustering process, rather than employing some time-consuming strategy typical of configurations described by figure 5.1.

However, the process preparation, not present in figure 5.1, does consume a time. The critical activities are finding the rules from GDN that match the goal of classification and the given context as well as searching the attribute domains. The process of searching the attribute domains is guided by the list of goal-relevant (attribute, value) pairs and is, hence, efficient. Searching the GDN for appropriate rules certainly depends on the number of rules. The more rules, the longer the search process. However, the more rules (as a result of the system's experience), the better the quality of the final result of the classification. Hence, a gain in the efficiency of the process means a loss in the quality of the result. The way out is a parallel evaluation of all the rules in GDN. There exists evidence,

supplied by the research in cognitive psychology, of the parallel processing capability of humans [Schneider and Shiffrin, 1977] [Shiffrin and Schneider, 1977]. Given the fact that the number of rules in GDN must be immense if a cognitive system is to deal successfully with the complexity of the surrounding environment, the parallel-processing assumption seems to be an imperative one. However, none of the approaches describable by figure 5.1 can be applied to a general domain, because of the amount of information necessary to be incorporated into the system and consequently the amount of time needed to process that information. They work in a specific domain of application with a limited amount of background domain-specific knowledge, and it takes a programmer to prepare it for application to another domain. If so, then the approach taken in this work, without the assumption of parallel processing but with a controlled size of GDN, offers an efficient alternative to the approaches described above. The size of GDN will influence the efficiency of the whole process in a directly-proportional manner.

The components of the classification process, described in figure 5.2, will be described in the following sections of this chapter.

5.2. Process Preparation

There are two main sub-processes within the process-preparation phase of the classification process:

- generation of the list of goal-relevant (attribute, value) pairs,
- climbing a domain hierarchy of the attributes that participate in the list of goal-relevant (attribute, value) pairs and the descriptions of given events.

5.2.1. List of Goal-Relevant Attributes

The first process to be performed is generation of the list of goal-relevant (attribute, value) pairs, that will serve as a driving mechanism for the clustering process. Given the goal of classification and context, the rules from Goal-Dependency Network are evaluated, and the ones that match the current description of the problem get a chance to participate in the process of classification. They add to the list of goal-relevant (attribute, value) pairs another pair (contained in the right-hand side of the rule) experientially relevant to the goal of classification. The relevance of the pair is essentially a copy of the strength of the rule posting the pair to the list. The more useful the rule has proved to be in past classifications, the higher the strength of the rule.

This is a proper place to explain the difference between the *strength* and *relevance* parameters. There are several strength parameters in the system: a strength of the hierarchical and associative links, a strength of the GDN rules, a strength of the heuristic criteria, etc. Instead of giving them different names and making the vocabulary rather complex, a common name *strength* is adopted for all of them, hoping that the context will disambiguate among them. The strength of each type of knowledge is calculated differently and the mechanism is explained throughout the study when needed. *Relevance*, on the other hand, represents the variability of features across a category and is of crucial importance for class descriptions (concepts), similarity measure, and the retrieval process. The different nature of the relevance parameter and its importance is emphasized by labeling it with a name not shared by other parameters in the system. The way of calculating a relevance is explained in one of the sections of this chapter.

5.2.2. Climbing a Domain Hierarchy

Once the list of goal-relevant (attribute, value) pairs has been created, the system can proceed with the second phase of the process preparation. This second phase is responsible for “cleaning up” the list of goal-relevant attributes as well as making the descriptions of given events as general as possible. The whole process is driven by the generated list of attributes, which presumably defines the level of abstraction of the result of classification. That assumption is based on the fact that the rules from GDN, selected to add a (attribute, value) pair to the list, match the goal and context of classification. But, that is not enough to make the claim valid. What we really need is to propose a form of rules in GDN that will make the assumption obvious.

The form of GDN rules has been described in 2.5.1. It is reprinted here in a slightly different arrangement:

```

if      goal1
          goal2
          ...
          goali and
          context1
          context2
          ...
          contextj

then    add to the list:
          (attribute, value, strength)

```

There is no restriction on the number of goals and attributes describing the context. But, there is a restriction on the content of both the left and right-hand side of a rule. Both sides must be as *specific* as possible. In the case of the left-hand side, it means that both the goal and context of classification must be described in as many details as possible. The reason for that is that we want the rule to be applicable to the appropriate situations only, without unjustified over-generalizing. People are hesitant to form general rules that would be applicable

to a whole range of situations, without great success in any of them particularly. If there were enough reasons to form a general rule, people would still try to apply exceptions to that rule whenever possible. In the case of the right-hand side of a rule, the specificity requirement is related to the structure of the attribute domain. What we want is to find the most specific value from the domain, still covering all known values that the attribute has used to take in the situations described by the goal and context part of the rule. As a result, a detailed description of the goal and context of classification and the most specific value of the attribute covering all known examples will determine the level of abstraction of the result of classification most appropriate to the task at hand. Consequently, the rules with these properties make the assumption stated above reasonable.

The list of goal-relevant (attribute, value) pairs then sets the upper bound on the level of generality of the attribute values used to classify given events with respect to the goal of classification. This fact can be used to guide the following subtasks of the process-preparation phase of the classification process:

- *Infer and add to the description of given events the (attribute, value) pairs from the list of goal-relevant attributes.* It may happen that the value of a specific attribute from the list is at a higher level of the domain hierarchy than the value of the same attribute in the description of an event from the set of given events. Then, it is obvious that we can infer the value from the list in the description of the event. The relevance will be the relevance of the lower-level value. There is no need to remove the lower-level value from the description of the event once the higher-level value has been added to the description. The reason is that, because of the choice of prototype as a representational mechanism, addition of a new (attribute, value) pair to the description of the event increases the similarity of the event with all concepts having the higher-value in the description, while keeping the same

similarity with the concepts having the lower-value in the description. This strategy will help us generalize the descriptions of given events to the level of abstraction defined by the list of goal-relevant attributes. The whole process is controlled by the *background knowledge* describing the structure of the domain. The search space is significantly reduced by the guiding value supplied by the list of goal-relevant attributes.

- *Remove the (attribute, value) pairs, from the list of goal-relevant attributes, covered (which lie at a lower level of domain hierarchy) by some other pair in the list.* It is quite possible that GDN rules produce several values at different levels of the domain hierarchy for the same attribute. Having assumed that the (attribute, value) pairs in the list of goal-relevant attributes define the appropriate level of abstraction for the classification process, and that the pairs in the list are as specific as possible, it is reasonable to suggest a removal of all (attribute, value) pairs that are more specific than some other pair for the same attribute.
- *Remove the (attribute, value) pairs, from the list of goal-relevant attributes, with the relevance below the prespecified threshold.* This is not to be confused with the threshold consulted during the process of evaluating the rules from GDN. That threshold is a result of the experience the system gained during the constant interaction with the environment. The value of the threshold used in this phase of the process preparation depends on the value of the (attribute, value) pair with the highest relevance in the list of goal-relevant attributes. The higher the highest relevance, the higher the value of the threshold. But before possible removal, we want each pair in the list to give its full contribution to the previously described subtasks of the process preparation. In general, the main reason for removing a pair from the list at all is *efficiency* of the clustering process. The fewer the (attribute, value)

pairs in the list of goal-relevant attributes, the smaller the classification tree traced by the clustering process.

- *Remove the (attribute, value) pairs, from the list of goal-relevant attributes, not covering any of the given events.* Obviously, these pairs would not make any contribution whatsoever to the clustering process, and, as such, should be disregarded.

After the process-preparation phase of classification is over, resulting in enriched descriptions of given events and a reduced list of goal-relevant attributes, the clustering process may take over control.

5.3. Clustering

There are three subtasks within the clustering process:

- utilization of the goal and current context of classification,
- utilization of the goal and common context of classification,
- utilization of the associative links of given events.

Each of them is explained in greater detail in the coming sections.

5.3.1. Utilization of the Goal and Current Context

This section can be further subdivided, as well, to

- evaluation of the list of goal-relevant attributes,
- best-first search,
- evaluation of the resultant clustering,
- GDN rules and threshold update.

5.3.1.1. Evaluation of the List of Goal-Relevant Attributes

The list of goal-relevant attributes contains (attribute, value) pairs supplied by the GDN rules whose left-hand side matches the goal and current context of

classification. The relevance of (attribute, value) pairs reflects the strength of the rule that posted the pair to the list. The higher the relevance, the more useful the rule has proved to be in the past, and, consequently, the more relevant the (attribute, value) pair is expected to be to the process of classification. As a result, the first attribute to be used in classification is the one with the highest relevance. However, the system will test the given events against not only the value with the highest relevance, but all the values (for that attribute) found in the list as well. Each value effectively defines a class of events. All values taken together form a set of disjoint classes. The question is, then, do they cover all the events from the input set? The answer is *no*, for there may be events that have no value for that attribute at all, or have some value other than the ones contained in the list of goal-relevant (attribute, value) pairs. Consequently, the system has to generate an additional class that will account for such events.

Another strategy could be employed as well, the strategy of testing the set of input events on one (attribute, value) pair at a time. Obviously, that pair should be the one with the highest relevance not yet tested. At each level of the classification tree two classes would be generated, the class of events with that property and the class of all other events from the input set. This approach has three main disadvantages compared to the approach described above:

- 1) The whole process is much less efficient since the complete computation would be repeated not once per attribute (as adopted in the approach described in the study) but once per value of each attribute. (The *depth* of the classification tree is much greater).
- 2) Consequently, a description of the path to a leaf of the tree is made unnecessarily complex and unintelligible, especially by the fact that one attribute may appear more than once in the path description.

- 3) There is no guarantee that the final result would be of any better quality than the one produced by the approach taken in this work to justify the inherent inefficiency.

5.3.1.2. Best-First Search

Once the system has the attribute (with corresponding values) to be used in the process of clustering, the actual clustering can take place. But, there is another question: should the system evaluate *all* the classes already generated or just *one* at a time? If one at a time, which one? It is obvious that at the beginning both approaches will produce the same result, since the classification process starts with one class only, the root of the classification tree. The *all-classes* approach will evaluate each (non-empty) node of the tree at all levels, thus building an almost complete tree. The problem with this approach, then, is that some nodes may be unnecessarily evaluated. As a result, the number of classes suggested by the classification process may be higher than necessary. And if for some reason the number of classes has been predefined, the problem is how to reduce the number of generated classes to the required one in an optimal manner.

The answer to those problems is to incrementally add as few newly generated classes as possible to the already existing ones. The other approach, *one class* at a time, will give us exactly that. In order to avoid an exhaustive search, the choice of a class to be evaluated, against the attribute from the list of goal-relevant (attribute, value) pairs with the highest relevance, is driven heuristically. Since people always try to further divide a class with the most members, the following heuristic reflects that observation: *the next class to be evaluated is the one with the highest cardinality*. (In the case of a tie, choose randomly). This heuristic is a logical consequence of the heuristic clustering-evaluation criterion which favors clusterings with roughly equal distribution of events across classes (initially described in Chapter 2, still to be discussed in this chapter).

Since at each level of the classification tree only the (heuristically) *best* candidate node gets evaluated, the search for the best clustering has the form of the *best-first* search. The maximal number of classes generated during each iteration of the process is equal to the number of values for the given attribute, supplied by the preceding phase of the algorithm. Before a class is generated, the system will check if there is any event satisfying the defining value of the class. If not, the class will not be generated. Also, the class with only one member does not get evaluated, since it cannot improve the heuristic clustering-evaluation criterion. The result of singleton-class evaluation is a new singleton class along with a class with no members.

In general, the *one-class* approach will be more efficient than the *all-classes* approach when both approaches generate a classification tree of the same depth. Otherwise, the efficiency will depend on the difference in the depth of two trees as well as the number of generated classes. Figures 5.3 and 5.4 describe these two approaches to clustering on the same example from the domain of UNIX user commands (the example will be explained in greater detail in Chapter 7). The commands to be classified are given in the root node of the classification trees. The attribute to be used as a basis of clustering at a specific level is given in bold font as a label on the left-most arrow pointing to a value of that attribute. The attribute value is written inside a circle. The value "other" means *other than values of the attribute specified in the list of goal-relevant attributes*. A solid box (including the root node) represents a class, defined extensionally by the commands written in the box, still to be evaluated. A dashed box represents a class in its final form. Note that, in this particular example, six (6) nodes (classes) have been evaluated by the all-classes approach and five (5) by the one-class approach.

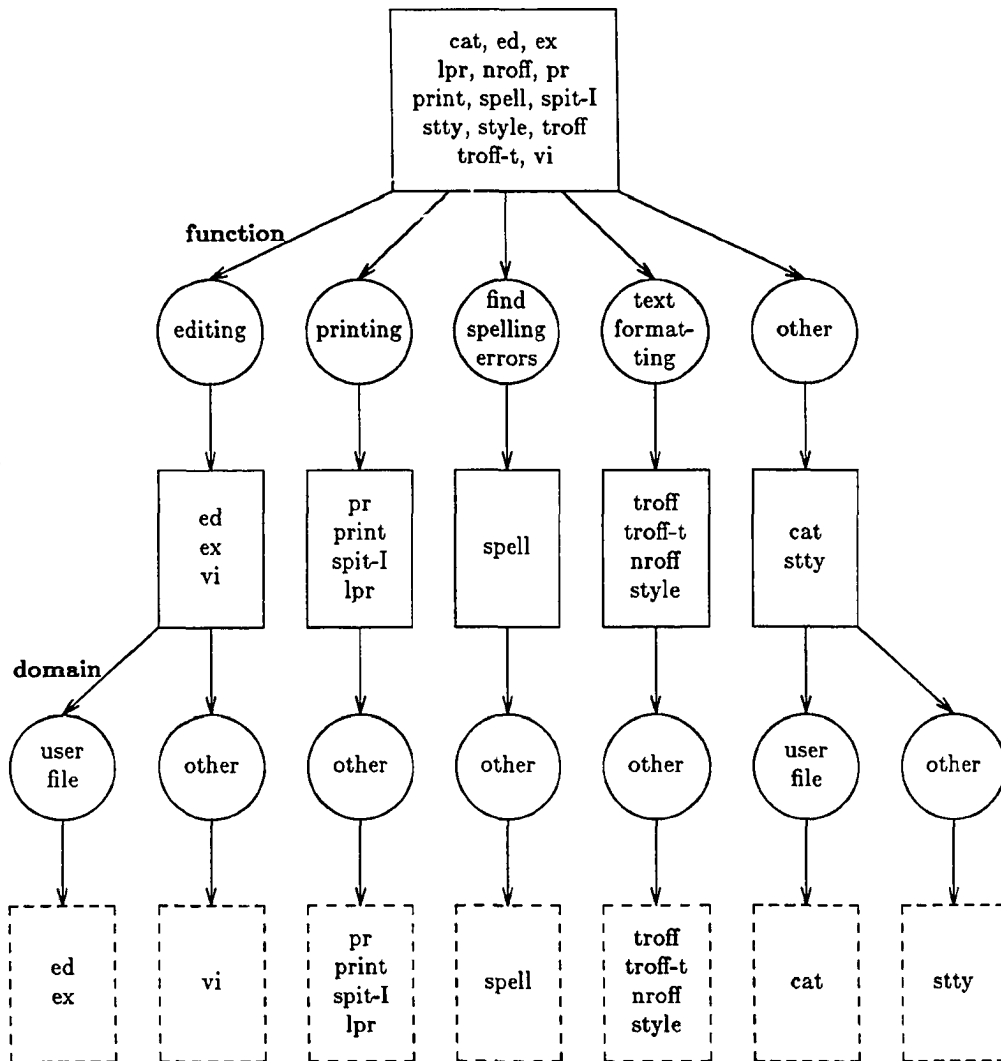


Figure 5.3 The *all-classes* approach to clustering.

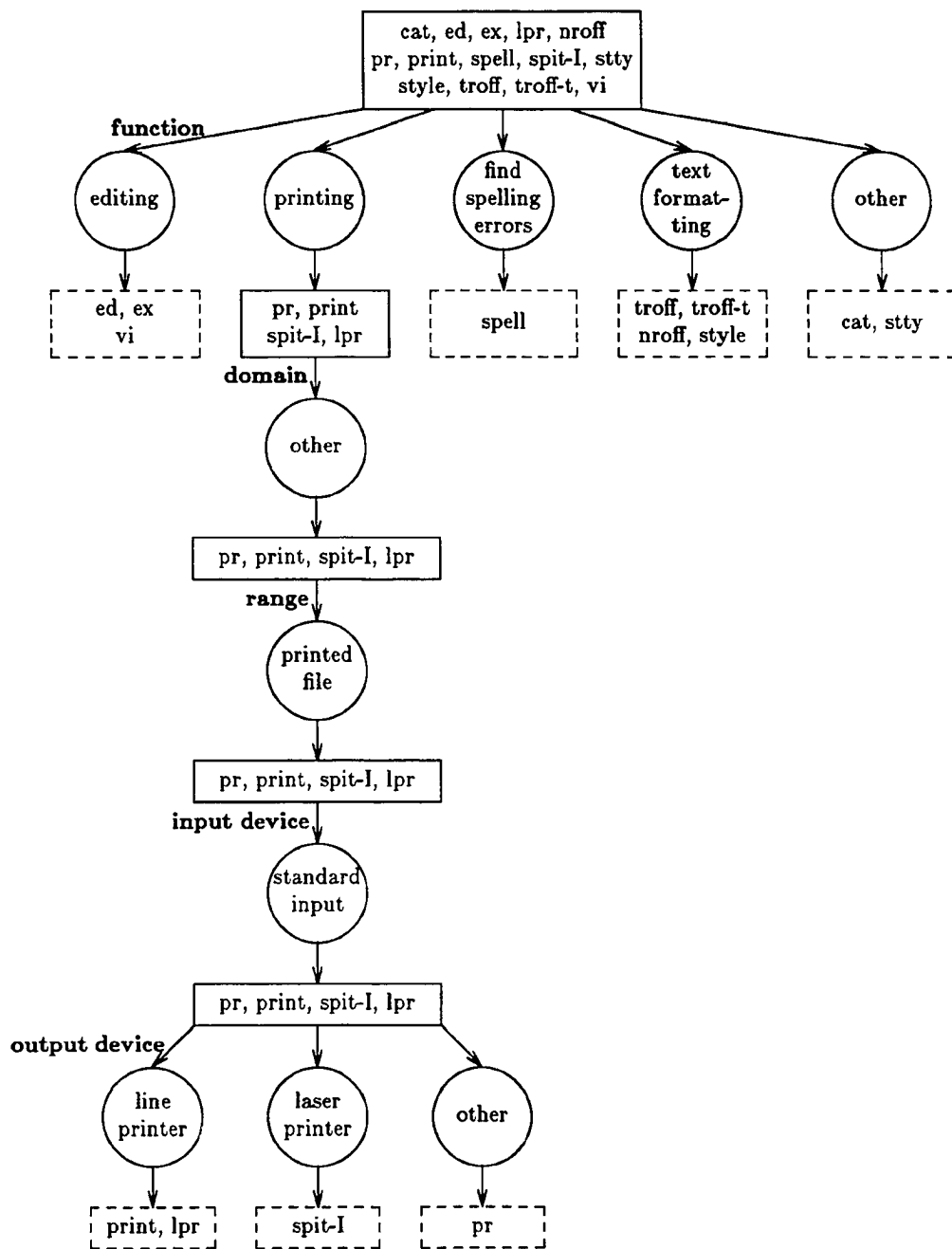


Figure 5.4 The *one-class* approach to clustering.

In the case of a predefined number of classes, if more than the requested number have been generated, the *one-class* approach will be more efficient since the system knows the evaluation of exactly which class has caused the problem. To correct the situation, two newly generated classes with the lowest cardinality should be merged and so on, until the desired number of classes has been achieved.

For the sake of comparison, a brief description of the *search control* mechanism employed by other authors in the clustering phase of their conceptual clustering algorithms is given. None of the authors referred to in Chapter 1 have systematically considered all possible groupings. As pointed out by Fisher and Langley [1985], CLUSTER/2 uses a hill-climbing method (*seed selection*) to generate an acceptable clustering, employing characterization techniques to evaluate it. The remaining systems (RUMMAGE, DISCON, etc.) carry out only a *degenerate* search, selecting the clustering in a one-step process. The reason for this one-step search is that, for example, RUMMAGE and DISCON both require a user-specified list of attributes and their values, and, consequently, by selecting an attribute, these systems automatically generate a candidate clustering. The same effect in a more data-driven manner is accomplished by GLAUBER, MK10, and UNIMEM/IPP.

The system defined by the approach accepted in this work, on the other hand, does not assume a user specified list of attributes and their values. It interacts with GDN in the process of creating a list of goal-relevant attributes. It then employs the process-preparation procedure along with the knowledge of the structure of attribute domains to modify the list of attributes as well as descriptions of given events. From that point on, a heuristically-driven one-step search process is employed. The evaluation process of the resultant clustering is explained in the following section.

5.3.1.3. Evaluation of the Resultant Clustering

The reasons for introducing the *heuristic clustering-evaluation criteria* and the comparison with *LEF with tolerances* [Michalski and Stepp, 1983] have been given in Chapter 2. Also, one of the elementary criteria has been identified: *a distribution of events across the generated classes should be roughly uniform*. The term *roughly uniform* is left to be defined empirically and in this work is given as a coefficient multiplying the average cardinality of the produced clustering. The result of clustering has to satisfy all the elementary criteria playing the role of the heuristic clustering-evaluation criterion (essentially the elementary criteria with strength above the dynamically defined threshold - fully described in Chapter 2).

A user supplied number of classes to be generated can guide the process of clustering as well. In the preceding section, the advantages of the *one-class* approach to clustering in that aspect have been discussed.

It is worth mentioning, however, the importance of the clustering-evaluation criteria. The more suitable the criteria to the problem domain, the better the quality of the result of classification. The elementary criteria should be as specific as possible, still covering all the problem domains they were intended to cover. Both too-general and too-specific descriptions of the criteria would cause inadequate evaluation of clusterings. A too-general description would recommend an acceptance of many bad clusterings, while a too-specific one would cause a rejection of many good clusterings. Either case is not acceptable when the goal is a psychologically plausible classification.

5.3.1.4. GDN Rules and Threshold Update

Once the clustering has satisfied the heuristic clustering-evaluation criterion, the system may continue with the characterization phase of the classification pro-

cess. But, if the clustering failed to satisfy a given criterion, the system is ready to employ the expectations-generation process, and try again. However, before proceeding with either one of the proposed actions, the system should utilize the information made available by the process of a clustering evaluation, and update its confidence to the participating GDN rules and corresponding threshold (a threshold defining the minimal strength a GDN rule should possess in order to compete for participation in the classification process).

Next, an analysis of both cases is provided and a mechanism of update outlined. If the clustering is successful, the system performs the following actions:

- + Increase the strength of GDN rules that have made a contribution to the clustering process. A rule has made a *contribution* to the clustering process if it has posted (to the list of goal-relevant attributes) an attribute, along with the corresponding value, that helped the system to reduce the difference between the *minimal* cardinality of the newly generated clustering and the *acceptable* cardinality, as defined by the heuristic clustering-evaluation criterion. The assumption is, of course, that the minimal cardinality is lower than the acceptable one. An example of GDN rules that should have the strength increased are the rules that have posted the values for the attributes *function* and *output device* in the example shown in figure 5.4. The actual increment $\delta_1(r_i)$ of the strength of the rule r_i is defined as:

$$\delta_1(r_i) = \frac{1}{l + k_1} \times s(r_i) \quad (5.1)$$

where l is defined as a number of pairs in the list of goal-relevant attributes (assuming that the *process preparation* is over, thus having taken into account the events description, attribute domains structure, and the value of the GDN rules threshold), k_1 is a positive integer whose value is to be empirically determined, and $s(r_i)$ represents a strength of the rule r_i . The rationale

behind this form of increment is the following: the higher the number of the (attribute, value) pairs in the list, the better the system is equipped to deal with current and similar situations; consequently, don't change the balance between the rules too radically, since the rules have proved themselves in the past. k_1 should not have a significant influence in this case. On the other hand, k_1 has the important role of balancing the possible negative influence of l when it gets too small. If there are too few (attribute, value) pairs in the list, the system, because of a lack of experience, tends to over-estimate the usefulness of the specific pair in future similar situations. k_1 is expected to curb those cases effectively. As a result, a small value of k_1 is suggested (i.e. $k_1=2$). Finally, $s(r_i)$ parameter makes all changes relative to the actual strength of the rule.

- + Increase the value of the GDN rules threshold. The rationale: since this is *one more* successful attempt, the system gets more and more confident in its performance - the system's action becomes a routine. To become a routine, it must be cleaned of all unnecessary steps. By raising the level of the threshold, the system effectively reduces the number of (attribute, value) pairs that will be used for classifications in future similar situations. If the system gets over-confident and suffers several failures, a similar mechanism (to be explained later in this section) will be activated to bring things back to normal. The actual increment δ_2 of the strength of the threshold has a form similar to the one given for the increment of GDN rules strength:

$$\delta_2 = \frac{1}{l + k_2} \times t_{GDN} \quad (5.2)$$

where k_2 , a corrective coefficient, behaves similarly to k_1 , and t_{GDN} represents a value of the GDN rules threshold.

- Decrease the strength of the GDN rules that have not made a contribution

to the clustering process. Those are the ones that have not reduced the difference between the acceptable and minimal cardinality of the clustering. A computation of the actual value of decrement for the strength of the corresponding GDN rules is equivalent to 5.1. An example of such rules are the ones that posted values for the attributes *domain*, *range*, and *input device* in the example shown in figure 5.4.

Otherwise, the following actions are performed by the system:

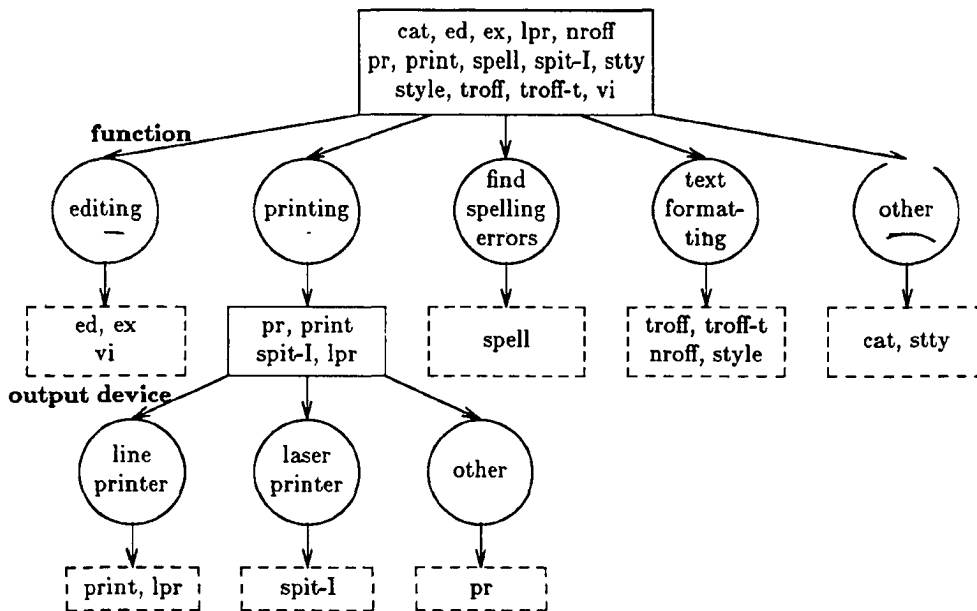
- + Increase, according to 5.1, the strength of the GDN rules that have made a contribution to the clustering process, but with no success. These rules are not responsible for the lack of other GDN rules that could have made the clustering successful, had they had a strength above the threshold or had they been known to the system at all.
- Decrease t_{GDN} by δ_2 . With this action the system may next time, in a similar situation, give a chance of posting a (attribute, value) pair to the GDN rules that would have added a pair to the list of goal-relevant attributes at this occasion had their strength been higher than t_{GDN} .
- Decrease, as defined in 5.1, the strength of the GDN rules that have not made a contribution during this unsuccessful attempt. The system should be able to learn from its mistakes.

In conclusion, there are two observations that should be made about the nature of the implemented update mechanism:

- The mechanism preserves the quality of the result of clustering, while improving the efficiency. It will make sure that the “good” rules are rewarded, while the others will not be consulted again in similar situations, after several consecutive failures to make a contribution to the clustering process. If we go back to figure 5.4, after several sessions we would expect a system to produce the classification tree as shown in

figure 5.5.

- The mechanism is rather *conservative*. It is based on the observation that a *setback* counts more than a *successful* attempt, when it comes to people's everyday behavior. People prefer stability of performance to the possibility of unpleasant surprises. Consequently, after several consecutive failures and the same number of subsequent successes (assuming the same value of l parameter), the system will not get back to the same value of the threshold (or GDN rule strength) it began with. The resulting value will be somewhat lower than the initial one.

**Figure 5.5** An example of improved performance of the clustering process.

5.3.2. Utilization of the Goal and Common Context

According to the *visibility* parameter, the information provided by the expectations-generation process is to be consulted next. The expectations-

generation process will supply additional information, as to what properties of the environment are likely to hold in the current situation, even though not explicitly mentioned in its description. This additional information may cause several new GDN rules to fire, thus supplying new (attribute, value) pairs to the list of goal-relevant attributes. This extended list (new + old pairs) will guide the clustering process in the next iteration. Once the list of goal-relevant attributes has been generated, the rest of the process is identical to what has been described in the previous section; hence it will not be repeated here.

5.3.3. Utilization of the Associative Links

If none of the previous two subphases of the clustering process have produced satisfactory clustering, the system can still try one more source of information: the associative links of the given events.

5.3.3.1. Evaluation of the Given Events' Associative Links

The system evaluates the associative links of the set of given events by keeping statistics on how many times a specific event (not necessarily from the input set) has been pointed to by events from the input set. Also, the system will compute a cumulative support $cs(e_i)$ for such an event e_i as:

$$cs(e_i) = \sum_{j=1}^n s_a(e_j, e_i) \quad (5.3)$$

where e_j is an event from the set of given events G , i.e. $e_j \in G$, a cardinality of the set G is n , and $s_a(e_j, e_i)$ represents the strength of the associative link between the events e_j and e_i .

An event that is associatively related to more events from G than any other event gets a chance to add all (attribute, value) pairs from its description to the list of goal-relevant attributes. After the *process-preparation*

phase, the system is ready to repeat the clustering process once again.

If there is more than one event with exactly the same number of “votes” from the given events, then the cs (cumulative support) parameter is called upon to decide. The event with the higher score wins. If it happens again that the scores are identical, a winning event will be chosen in a random fashion.

5.3.3.2. Associative Links Strength Update

Once again, after the clustering process is over, the system will employ its update mechanism to learn from the last experience. If the clustering was successful, the system will increase the strength of the associative links of the given events which “voted” for the event that made the clustering successful. The increment $\delta_3(e_i, e_j)$ of the strength of the associative link between the events e_i and e_j is defined as:

$$\delta_3(e_i, e_j) = \frac{1}{v + k_3} \times s_a(e_i, e_j) \quad (5.4)$$

where v stands for the number of “votes” that the winning event has collected, and a corrective coefficient k_3 behaves similarly to k_1 and k_2 , previously described. The justification for 5.4 is similar to the justification given for 5.1. When the value of v is high, it is obvious that many events had known about that specific event, and its high score came as no surprise. The amount of information gained by the system is small. When the opposite is true, i.e. the value of v is low, the system gained a significant amount of information. The increment of the strength of an associative link should reflect these observations. The k_3 parameter will consequently prevent the system from making some radical changes in the event descriptions after seeing just a few instances of a specific situation.

However, if the clustering was not successful, the system will decrease the strength of the associative links of the given events which voted for the event blamed for the failure of the clustering process. The actual decrement of the strength of each specific link will be calculated according to 5.4.

5.4. Characterization

The characterization process follows successful clustering. This phase corresponds to *learning from examples*. Given a set of instances of the class, the problem is to generate the *most specific* description of the class, covering all of the presented instances and as few as possible of the instances not belonging to that class. According to the discussion presented in Chapter 4, the first step in the characterization process is generation of the prototypical description of the class.

5.4.1. Generation of a Prototypical Description

All (attribute, value) pairs used in the descriptions of given events should be represented in the class description. The more (attribute, value) pairs in the class description, the more specific the description, for an event must match more attributes and their values to claim membership. Different aspects of *similarity* will be discussed later in this chapter.

According to the approach taken in the *process-preparation* phase, some attributes may have more than one value in a class description. In fact, every time the specific property is not shared by all the members of the class, the class description will contain multiple values for the corresponding attribute. If one value of the attribute covers all other values, it would be easy to clean-up the description, much like the system did in the process of cleaning-up the list of goal-relevant attributes. However, what happens when that is not the case? How far should the system climb the attribute

domain hierarchy to reach the node that covers all the values? How general is newly generated value? Is it too general to be of any use in dealing with the task at hand? There are no simple answers to these questions.

The approach to this problem, accepted in this work, is based on the assumption stated earlier: the list of goal-relevant (attribute, value) pairs, supplied by the GDN rules matching the goal and context of classification, determines the level of abstraction at which the result of classification should be generated. If that is so, then the system has reached that level of abstraction during the process-preparation and clustering phase of the classification process. The list of goal-relevant attributes has been cleaned up, the event descriptions have been enriched, and the clustering has been guided by the list itself. The bottom line is that any further attribute domain hierarchy-climbing would over-generalize the class description, which is exactly what the system should not do. Consequently, approach taken in this work points out that the characterization process should take into consideration all candidate (attribute, value) pairs for the class description, and filter out only the ones not shared by at least half of the members of the class (another heuristic). In the next section the process of computing the relevance of an (attribute, value) pair from the class description will be described. At that point, it will be possible to state an even stronger criterion.

The *multi-value* approach to class description can give us some additional information about the class members as well. For instance, if the class description contains the *(function, file-manipulation, 0.87)* and *(function, text-formating, 0.75)* attribute-value-relevance triplets, one possibility is to remove the latter triplet since the value *text-formating* is covered by the value *file-manipulation* (at least according to the description of the *func-*

tion domain adopted here). However, if the system lets both triplets participate in the class description, then it can derive the following information about the members of the class: most members have the function of manipulating files, but the majority of them are specialized in text formatting. This kind of information may prove to be useful in a specific context.

5.4.2. Attribute-Value Relevance Calculation

The relevancy rel of a (attribute, value) pair (att, val) in the description D of the class c is computed as a sum of relevances of the class members with that particular property, divided by the total number of members, i.e.:

$$rel(c, att, val) = \frac{\sum_{i=1}^m rel(e_i, att, val)}{n} \quad (5.5)$$

where $e_i \in c$, $(att, val, rel) \in D_a(e_i)$, $D_a(e_i) \subset D(e_i)$, $D(e_i)$ is a description of the event e_i , $D_a(e_i)$ represents a subset of $D(e_i)$ containing the (attribute, value) pairs only (no hierarchical or associative links), n is the cardinality of the class c , and $m \leq n$ stands for the number of events $e_i \in c$ with the property (att, val) .

It was mentioned previously that a (attribute, value) pair will be “accepted” in the class description if it is true of at least half of the members of the class. Now the same criterion can be stated in terms of the relevance of the (attribute, value) pair, i.e., if the relevance of the corresponding pair is greater than or equal to 0.5. However, this criterion is more restrictive than the previous one. The relevance of 0.5 will stand for half of the members if, and only if, that half of the members has the respective property with the relevance 1.0. Otherwise, more than half of the class members have the specific property when the corresponding relevance is 0.5. The value of this threshold is to be determined empirically. However, it is in order to mention that there is a trade-off between the overall efficiency and

specificity of the generated concepts. The higher the value of the threshold, the fewer the (attribute, value) pairs in the class description. The fewer the elements in the class description, the more general the description, since more events have a chance to match the description. Also, the fewer the (attribute, value) pairs in the class description, the more efficient the process of building a hierarchy and retrieval.

One of the by-products of the implemented mechanism of relevance calculation is the property of *inheritance*. It is implemented in a bottom-up manner, much the same way as people build their knowledge of categories through experience with instances. At any level of a hierarchy, a concept will have the relevance *1.0* for the (attribute, value) pair (*att, val*) if, and only if, the same property is shared by all the members of the class and with the same relevancy, i.e. *1.0*. Consequently, whenever the system comes across a (attribute, value) pair with the relevance *1.0* in the description of a class, it knows that the same feature must be true of all of its members.

5.4.3. Hierarchical Links Strength Calculation

Once the class description has been generated and the corresponding relevances calculated, the system can proceed and compute the strength of the hierarchical links: *specialization-of*, *generalization-of*, *instance-of*, and *has-instance*. The same mechanism is employed to calculate the strength of all types of hierarchical links. When trying to determine the degree of match between a concept and an instance, for example, the system will sum up the products of relevances for the attributes that have the same value in the descriptions of both the class and event. The resulting sum is then divided by the number of (attribute, value) pairs (no links) in the class description. In the class description this value will be introduced as the strength of the *has-instance* link, while the event description will have the same strength for

the *instance-of* link. The same discussion holds in the case of the match between a concept and its sub-concept. The only difference is that the names of the links in this case will be *generalization-of* and *specialization-of*, respectively. The strength-calculation process is summarized in the following equation:

$$link(c, cm_j) = \frac{\sum_{i=1}^n s(c, att_i, val) \times s(cm_j, att_i, val)}{n} \quad (5.6)$$

where $link(c, cm)$ stands for the strength of the link between the class c and its member cm_j , i.e. $cm \in c$, $s(c, att_i, val)$ represents the strength of the (att_i, val) pair in $D_a(c)$, $(att_i, val, s) \in D_a(c)$ and $(att_i, val, s) \in D_a(cm)$, and n is a cardinality of $D_a(c)$.

It is interesting to point out that if we introduce the threshold to filter out all the links with the strength below it, we have effectively implemented a mechanism for forgetting “not so important” relationships. The value of the threshold should be determined empirically. The higher the value of the threshold, the fewer the nodes in the classification tree (a tree pruning). Consequently, although the retrieval process gets more efficient, the result of the retrieval process becomes less complete.

5.4.4. Associative Links

Associative links, in general, can be updated but not created by the classification process. They are created as a result of the system’s experience through activities of other cognitive processes. As a result, the associative links that a member of a class may have do not get “delegated” to the class description. However, similarly to the threshold introduced for hierarchical links, we can define a threshold which would cause the associative links with a low strength to be dropped from the event’s description.

5.4.5. Similarity Function

At this point, a *similarity function* can be defined simply as $link(c, e)$ defined by 5.6. It is a *degree of match* between the class c and the event e . To see which of the generated classes c_i the new instance e belongs to, the system should compute $link(c_i, e)$ for $i=1, \dots, n$ where n stands for the number of classes under consideration. The event is assigned to the class which maximizes the value of $link$. Table 5.1 summarizes the value of $link$ between the UNIX command *ed* and the generated classes described in figure 5.4. The example itself is fully described in Chapter 7.

Table 5.1

class	<i>link</i>
ed, ex, vi	0.516
print, lpr	0.188
spit-I	0.25
pr	0.25
spell	0.273
nroff, style, troff, troff-t	0.146
cat, stty	0.184

However, this is not the only possible strategy. One of the alternatives would be to define a threshold value, and assign an event to all classes that score above the threshold on *link* parameter. This strategy effectively implements the concept of *intersecting categories*. Obviously, we would have to be careful in assigning a value to the threshold, for it would have a direct influence on the size of the set of intersecting categories.

In one of the previous sections the mechanism of calculating the relevance of a (attribute, value) pair in the class description has been discussed. It was mentioned that the system would drop the pairs with the relevance below the given threshold. It is of interest for this discussion to analyze the impact of dropping a pair from the class description to the value of *link*. Dropping a (attribute, value) pair from the class description increases the chances of events without that pair in their description to score high on *link* parameter - a *generalization* step. If a specific event has the corresponding pair in its description, we can distinguish two cases: (1) taking the pair into consideration would increase the value of *link* - a *specialization* step, and (2) the opposite is true - a *generalization* step. Consequently, the value of the (relevance) threshold should be determined carefully with respect to the similarity function.

5.4.6. Other Approaches to Characterization

When searching the space of hypotheses (of class descriptions - concepts), in order to find one that covers the given events, one may search:

- from a specific hypothesis toward more general ones (*generalization*),
- from a very general hypothesis toward specific ones (*specialization*),
- in both directions, hoping to converge on the correct hypothesis (*version space* strategy [Mitchell, 1978]).

UNIMEM/IPP and GLAUBER employ generalization in characterizing their groupings. CLUSTER/2 uses a discrimination approach to derive a maximally-general discriminant concept and a generalization approach to derive a maximally-specific characteristic concept. RUMMAGE and DISCON employ a list of attribute values to form partitions, where each attribute

value represents a maximally-general discriminant concept of the corresponding event group (no generalization or specialization is employed in this process).

In terms of the method used to direct a search through the space of hypotheses, because of the limited representational mechanisms and languages employed by the conceptual clustering systems, there is exactly one maximally-specific class description for any given event-group [Fisher and Langley, 1985]. In other words, there is no search (or only a *degenerate* one) occurring in most cases. CLUSTER/2 differs from the rest of the discussed systems in the sense that it carries out a *beam* search in deriving maximally-general discriminant concepts, using an *LEF* evaluation criterion supplied by the user.

Finally, the conceptual clustering systems can be compared with respect to the nature of operators for moving through the space of hypotheses. CLUSTER/2, DISCON, RUMMAGE, GLAUBER, and MK10 require data to direct the search through the problem space. The remaining systems can be viewed as *model-driven* systems, although the models used by DISCON and RUMMAGE consisted only of a list of given attributes.

5.5. Building a Hierarchy

As opposed to learning from examples which is generally concerned with forming concepts at a single level, classification systems usually focus on generating a *hierarchy* of concepts. In the case of conceptual clustering systems discussed so far, the search for clusterings and the search for characterizations are embedded within a higher level search through the space of classification systems.

5.5.1. Description of the Implemented Approach

The implemented approach to the process of building a hierarchy is quite different. It assumes that the characterization process is the first step toward the hierarchy. The hierarchy-building module then takes the classification tree, generated through the process of clustering, and works its way up by invoking the characterization process at each level of the classification tree. The concepts at the level immediately below the current level of the hierarchy represent the sub-concepts to be taken as the basis of the characterization process. The process is over when the root node has been reached. The higher the level of hierarchy, the more general the concept. Its description will contain fewer (attribute, value) pairs, thus effectively covering more events.

The process is essentially data driven in nature. It begins with the result of the clustering process, at some intermediate level of generality, and climbs the hierarchy toward more general class descriptions. In some sense, the clustering process is a component of the hierarchy-building process as well. It traces the classification tree. However, the characterization process is not consulted at each and every step, but rather when the clustering process is finally over. Also, it is the clustering that gets evaluated in the approach presented here and not the class descriptions themselves. Since the hierarchy is considered to be a hierarchy of class descriptions (concepts), it was plausible to define the hierarchy-building module without the clustering process itself.

5.5.2. Other Approaches to Hierarchy Building

The majority of the conceptual clustering systems have used *divisive* (top-down) methods of hierarchy building, including CLUSTER/2, DISCON, and RUMMAGE. Divisive methods start with a single class of given events,

and proceed by subdividing the events into classes, sub-classes etc., until the system is satisfied with the final result. GLAUBER and MK10, on the other hand, use *agglomerative* (bottom-up) methods, which begin with separate classes for each event, merging the classes when justified, until they have reached the appropriate level of generality. UNIMEM and IPP, however, first form classes of medium generality, and later form both more general and more specific classes.

In terms of a *search control*, the CLUSTER/2, RUMMAGE, GLAUBER, and MK10 systems carry out only a degenerate search through the space of hierarchies. The reason is that these systems are interested in finding optimal clusterings and characterizations, hoping that they will extend the quality of their results to the hierarchy as well. In contrast, DISCON, UNIMEM, and IPP carry out a search at the level of a hierarchy. DISCON, for example, carries out the degenerate search at the lower levels, but employs a best-first search schema through the space of hierarchies.

5.6. Feedback from the Performance System

Once the classification process is over, the problem-solving process receives the result of classification, and resumes control of the system's behavior. However, this is not the end of the interaction between the classification and problem-solving processes with respect to this last experience. The system should learn as much as it can from past experience in order to improve its performance and adaptive power. Consequently, once the problem-solving process is done with the task at hand, it is responsible to get back to the classification component and provide the feedback on the successfulness of the result of classification in the course of solving the current problem.

Given feedback from the problem-solving process, the update of the heuristic clustering-evaluation criteria is in order. If the result of classification proved to be successful, the system would increase the strength s_h of the heuristic criterion h_i by $\delta_4(h_i)$ as defined in 5.7:

$$\delta_4(h_i) = \frac{1}{cl_{gen} + k_4} \times s_h(h_i) \quad (5.7)$$

where cl_{gen} represents the number of classes generated by the classification process, and k_4 is a corrective coefficient, similar in function to k_1 , k_2 , and k_3 . On the other hand, if the result of classification has not been helpful to the problem-solving process, the system will decrease the strength of the heuristic criterion for the same amount.

The rationale behind 5.7 is based on the following heuristic: the smaller the number of classes, the better the result of classification. It is based on the observation that people tend to classify events in as few classes as possible, in order to keep the level of details manageable. According to 5.7, the success or failure of the result of classification with a large number of generated classes will not change the strength of the heuristic criterion drastically. The situation is quite the opposite when the number of generated classes is small, causing significant changes in the value of the heuristic criterion.

Finally, it is important to point out that there are two levels of the update process, implemented in this work. The first level includes the update of GDN rules, thresholds, and associative links, upon the completion of the clustering process. It is made possible by the presence of *internal* feedback, provided by the heuristic clustering-evaluation criteria. The second level of updates is initialized by *external* feedback from the problem-solving process, after it has had a chance to evaluate the result of

classification in its intended environment. This level includes the update of the strength of the clustering-evaluation criteria. What is the relationship between the two levels? Since the updates at the first level are driven by the clustering-evaluation criteria, which are, on the other hand, updated at the second level, it is obvious that the updates at the second level influence the nature of the updates at the first level. As a result, the update process is driven by the performance system itself.

CHAPTER 6

CLASSIFICATION ALGORITHM

6.1. Algorithm

Having defined the model of cognitive systems, the classification process, and the chosen knowledge representation mechanism, it is time to define the algorithm of classification. The algorithm will certainly reflect all aspects of the analysis performed in previous chapters, combining them into a whole capable of solving a classification task at hand. The first section of this chapter will outline the skeleton of the algorithm, leaving the details of individual steps to be described in subsequent sections.

6.1.1. The Outline of the Algorithm

1. Prepare initial conditions for classification process.
2. Cluster the set of given events.

After each phase,

- 2.1. evaluate produced clustering with the heuristic clustering-evaluation criteria, and
- 2.2. update the strength of the source of information according to the result of evaluation.
3. If the clustering was not successful, engage the end user in a *help* dialogue.
Otherwise,
 - 3.1. Characterize generated classes.
 - 3.2. Derive a hierarchy of concepts based on the classification tree traced

out by the clustering process.

3.3. Evaluate the result of classification through the interaction with the performance system.

3.4. Update the strength of heuristic clustering-evaluation criteria accordingly.

6.1.2. Process Preparation

1. Add the goal of having evaluated GDN rules to the agenda. The process is to be carried out with respect to the goals and context of classification as well as the GDN threshold, thus effectively creating a list of goal-relevant (attribute, value) pairs.
2. Infer and add the (attribute, value) pairs from the list of goal-relevant attributes to descriptions of given events.
3. Remove “prohibited” (attribute, value) pairs, as well as the ones covered by them, from the list of goal-relevant attributes - given constraints.
4. Remove “prohibited” (attribute, value) pairs, as well as the ones covered by them, from descriptions of given events - given constraint.
5. Remove the (attribute, value) pairs covered by some other pair from the list of goal-relevant attributes.
6. Remove the (attribute, value) pairs with the relevance below the threshold, defined as a fixed percentage of the highest relevance, from the list of goal-relevant attributes.
7. Remove the (attribute, value) pairs not covering any of given events from the list of relevant attributes.

6.1.3. Clustering

1. Find the class with the greatest number of members (greater than 1) which has not been yet evaluated against all attributes from the list of goal-related attributes. In the case of a tie, choose randomly. Note that one class exists only in the initial stage, the one containing all of the given events.
2. Find the attribute with the highest relevance of all candidate attributes not used previously in the evaluation of members of the chosen class.
3. Generate a class (if non-empty) for each value of the attribute, plus the class of events having none of the corresponding attribute values.
4. Determine the members of the generated classes.
5. Evaluate the resulting clustering with respect to heuristic clustering-evaluation criteria.
6. If the resulting clustering fails to satisfy given criteria, repeat steps 1-6 until an appropriate class can be found.
7. Update the GDN threshold and strength of the activated GDN rules according to equations 5.1 and 5.2 of Chapter 5 and the result of the evaluation of produced clustering.
8. If this phase of the clustering process fails to produce a satisfying result, then
 - 8.1. Add the request for additional information from the expectations-generation process to the agenda.
 - 8.2. Repeat the process preparation phase of the algorithm, followed by steps 1-7 of the clustering module.
 - 8.3. If the heuristic clustering-evaluation criteria have not been satisfied yet, then

8.3.1.

Add the request for additional information from the spreading-activation process to the agenda. The process will evaluate the cumulative support for an event according to 5.3 and the set of given events.

8.3.2.

Repeat steps 1-6.

8.3.3.

Update the strength of the activated associative links according to equation 5.4 and the result of the evaluation of produced clustering.

6.1.4. Dialogue

1. Present the generated list of goal-relevant (attribute, value) pairs, along with the corresponding relevances, to the user.
2. Ask the user for suggestions with respect to plausible modifications of the strength of activated GDN rules and the GDN threshold.
3. Proceed with actual modifications, if any.

6.1.5. Characterization

1. For each class, generate a description containing all (attribute, value) pairs used in descriptions of member-events.
2. For each (attribute, value) pair of each concept, compute the relevance according to equation 5.5.
3. For each concept, remove (attribute, value) pairs with the relevance below the pre-set threshold.

4. For each event and concept, calculate the strength of hierarchical links according to equation 5.6.
5. For each concept, remove the hierarchical links with the strength below the pre-set threshold.

6.1.6. Building a Hierarchy

1. Get the classification-tree traced out by the clustering module.
2. Mark the leaf-classes on the longest path as new events to be characterized.
3. Employ the characterization process.
4. Remove these events from the classification-tree.
5. Repeat steps 2-5 until the root node has been reached.

6.1.7. External Feedback and the Update Process

1. Add the request for the evaluation of resultant classification to the agenda.
2. Update the value of heuristic clustering-evaluation criteria according to the result of the evaluation and equation 5.7.
3. If the result of the classification has proved to be successful with respect to the performance system, add the request for storing the resulting classification to the agenda.

6.2. Algorithm Analysis

There are several aspects of the algorithm that deserve to be discussed.

- Basic properties.
- Capabilities for incremental learning.
- Retrieval mechanism.
- Efficiency considerations.

- Possibility of semi-automatic generation of events representation.

Each of these topics will be analyzed to some extent in the rest of the chapter.

6.2.1. Properties

There are several properties that should be emphasized when describing the algorithm.

- The algorithm employs both the model-driven and data-driven strategy of search control. The clustering process is driven by the virtual and mental models in different phases of the process. The search process for the most plausible clustering is reduced significantly by both the list of goal-relevant (attribute, value) pairs, formed by GDN rules that match the goals and context of classification, and the associative links of the events to be classified.

The characterization process, on the other hand, is driven by the descriptions of input events, thus taking into account all features that may prove to be important for the class description.

The hierarchy-building process uses the classification-tree, generated by the clustering process, to guide the construction of the hierarchy. However, it will invoke the characterization process at each level of the hierarchy to describe concepts, thus effectively employing descriptions of input data to guide the process of class descriptions.

- The algorithm uses domain-specific knowledge to guide the general classification procedure. Domain-specific knowledge is embedded in the form of GDN rules and the description of the structure of attribute domains.
- The steps of the clustering process are ordered heuristically according to the visibility parameter. The more relevant the information to the goals and context of classification, the earlier it will be tested as a possible basis of the process of classification.

- The problem-solving process, representing a performance system, is accepted as the final “authority” to evaluate the result of the classification. If the problem-solving module is not implemented as a part of the system, then a user can play the role of the performance system.
- The algorithm provides a mechanism for learning from the past experience. After every session, based on the success or failure of the result of classification (determined by heuristic criteria and performance system), the algorithm will update the information listed below.
 - The strength of activated GDN rules.
 - The threshold of GDN rules.
 - The strength of the associative links of events to be classified.
 - The strength of heuristic clustering-evaluation criteria.
- The theory of prototypes provides a framework for the representational mechanism supported by the algorithm. The chosen representational mechanism enables the algorithm to account naturally for the properties listed below.
 - Intra-class feature variability.
 - Inheritance of properties true of all members of the class.
 - Existence of intersecting categories.
- All the properties of the algorithm listed so far increase the psychological plausibility of the result of classification. The clustering process is guided by the rules that take into account the goals and context of classification as well as properties of the events to be classified. Domain-specific knowledge helps the system to focus to the information relevant to the task at hand. The three phases of the clustering process are ordered according to the closeness of information used in the process to the problem description. The

problem-solving process evaluates the result of classification according to its usefulness to the process that requested it in the first place. The system learns from past experience, thus improving the quality of future classifications in similar situations. Finally, the representational mechanism has been adopted after consulting the results of the research on human representational mechanisms, conducted within the field of psychology.

6.2.2. Knowledge-Base Flexibility and Adjustability

An interesting question related to the classification algorithm is the one of the *flexibility* and *adjustability* of the concepts formed by the process of classification. This question includes several interesting topics.

- *Concept representation update.* Once the concept has been formed, the system needs to update its description appropriately after seeing another positive instance of the concept.
- *Noise handling.* How does a negative instance influence the quality of the generated description? How does the system prevent that from happening? In other words, what is needed is an algorithm that is robust with respect to the noise in the set of input data.

An interesting aspect of this topic is the question of imposing constraints (defined externally) on the concept description. Given a set of attribute values not to be taken into consideration, how does the system filter them out from the concept description?

- *Flexibility of generated concept descriptions.* Are the generated descriptions and the set of procedures operating on them flexible enough to provide a match with an instance whose description varies along several dimensions from the expected one?

The first two topics deal with the ability of the algorithm to learn incrementally, while the third one is concerned with the influence that the classification algorithm may have on the quality of the retrieval process.

6.2.2.1. Incremental Learning

A concept formed through the process of classification has taken into account a subset of the set of all positive instances of the concept. The question is then, what happens when a new instance has been encountered? Since the prototype theory suggests that a class is represented by its prototypical member (a real or an abstract one), it is obvious that the new member is likely to change the prototypical description. *What the nature of that change is* is the question that will be discussed in the following section.

6.2.2.1.1. Concept Representation Update

There are two possible ways the new instance may change a concept description: (1) add a new attribute value, and (2) modify the relevance of the existing (attribute, value) pair.

In order to support these changes in the algorithm, the system must have available the number of all instances of the concept seen previously (N_1). This is not an unrealistic requirement since people are able to recall pretty accurately (at least in a qualitative sense) the number of instances of the specific category they have seen until that moment. Why does the system need N_1 ? According to equation 5.5, to calculate the relevance of a (attribute, value) pair in the class description, the system needs to know the cardinality of the class.

So, if the system wants to add a new (attribute, value) pair to the class description, the only thing to be done is to compute its relevance. The relevance can be computed easily by dividing the relevance that the (attribute, value) pair had in the event's description by $N_1 + 1$. However, there is a problem. A threshold

filtering out the (attribute, value) pairs with low relevance from concept descriptions has been introduced. The problem is that the system might have had a chance to add this “new” (attribute, value) pair to the concept description already, but refused to do so because of the low relevance of the pair. What that means is that the computation of the pair’s relevance should take into account the relevance of the same pair that occurred earlier. With this newly added value of relevance, the (attribute, value) pair has a better chance of scoring above the threshold, thus becoming a part of the concept’s description.

One way of remembering old (attribute, value) pairs, once removed, is not to remove them at all, which can be achieved easily by assigning the value 0.0 to the corresponding threshold. There are two aspects of that decision, a positive and a negative one. The positive one is that any (attribute, value) pair can be added to the concept description if there is an instance with that property. The negative aspect of the decision is that the increased number of (attribute, value) pairs will cause the processing and memory efficiency of the classification process to drop significantly. The way out is obviously a trade-off between the two opposite requirements. The value to be modified is the value of the threshold. The lower the value of the threshold, the worse the efficiency of the classification process. But, the lower the value of the threshold, the more complete the description of the concept. As a result, the value of the threshold should be determined on the basis of the context and domain of application.

Another way of remembering old (attribute, value) pairs, after removing them from concept descriptions because of the low relevance, is storing them in a kind of *auxiliary* storage. The auxiliary storage would keep the information on the number of occurrences of a pair along with the sum of their relevances. Every time there is a new attribute value to be added to the concept description, the system would check the auxiliary storage for possible previous occurrences of the

same pair. If there is enough evidence that the pair in question should be added to the concept description, the system would proceed to do so. Otherwise, information about the new pair would be added to the auxiliary storage as well. This approach allows the system to retain *partially* its efficiency (it will take time to search the auxiliary storage), while still having the capability of adding new (attribute, value) pairs when justified.

The same threshold in the algorithm implemented in this work has been set to 0.5. The decision was guided by the following heuristic: *the feature should be true of at least half the members of the class to be represented in the class description*. The concept of auxiliary storage can be implemented even more efficiently in this case, because of the fact that the information on the number of previous occurrences of the pair would be sufficient to quit further testing most of the time (at least a half of the class members should possess the feature before it can be considered a candidate for participation in the class description).

The situation is much simpler when the goal is to modify the relevance of the already existing pair, after seeing an instance with the same property. The relevance can be increased, decreased, or remain at the same level. If it falls below the threshold, the pair gets removed from the concept description. Otherwise, it continues to participate in the description, but with the new value of the relevance factor. There are three possible strategies for updating the relevance of an (attribute, value) pair.

- The *conservative strategy* - past experience carries more weight than the changes hinted by the new appearances of the same (attribute, value) pair. Thus the system reacts rather slowly to the changes in the surrounding world. It doesn't always act in the best possible way, but, on the other hand, makes few mistakes.

- The *radical strategy* - the emphasis is placed on the shift in the value of the relevance. The past experience doesn't carry as much weight as the current trend in the behavior of the environment. The system reacts promptly to the changes in the outside world, but makes more mistakes in the long run.
- The *moderate strategy* - each appearance of the (attribute, value) pair carries the same weight. The system believes equally in the past and the present in its attempt to generate plausible expectations about the future. It reacts faster than the system employing the conservative strategy to the changes in the environment, while making fewer mistakes than the system employing the radical strategy. This is the strategy that is implemented in this work.

In order to implement any of these strategies, the system must know the number of events (N_2) with a specific property. This information will be used by the system to calculate the sum of the relevances recorded previously, which will, in turn, be used in the calculation of the new value of the relevance factor for the (attribute, value) pair in question. The actual method of computation of the new value of the relevance is described by the equation 6.1.

$$rel(c, att, val) = \frac{rel_p(c, att, val) \times N_2 + rel(e, att, val)}{N_1 + 1} \quad (6.1)$$

where $rel_p(c, att, val)$ stands for the sum of the previously recorded relevances of the (att, val) pair in the description of the concept c , and $rel(e, att, val)$ is the relevance of the same pair found in the description of the newly encountered instance e of the concept c .

6.2.2.1.2. Noise Handling

Noise handling is a very important property of any classification algorithm. It is always possible that an event would get misclassified into a wrong class, especially in the case of concept formation where there is no teacher to

distinguish between the positive and negative instances of a concept. Based on the algorithm presented in the first part of this chapter and the analysis from the preceding section, it can be stated that the approach to classification presented in this work has the property of decreasing *gradually* the influence of a misclassified event on the concept description.

This is going to be explained in detail. There are two kinds of (attribute, value) pairs the misclassified event can “delegate” to the concept description: (1) the pairs *not* shared by the rest of the class members, and (2) the pairs *shared* by other members of the class.

Because of the adopted value (0.5) of the concept-description threshold, the pairs not shared by the rest of the class members will not be a part of the description of the class if the class has more than two members. If the class has no more than two members, then the case is similar to the second one described above.

In the case of the (attribute, value) pairs shared by the other members of the class, two situations can be distinguished. The first one includes the pairs with the resulting relevance below the pre-set threshold. They get removed from the class description. The other situation describes the opposite case when the pairs stay in the class description with the relevance above the threshold. This is an interesting situation, and the term *decreasing-gradually* is actually defined in this context. The misclassified event adds its contribution to the resulting relevance. Consequently, the relevance may be either increased or decreased. But, whatever the case, the new events that represent positive instances of the concept will decrease the influence of the misclassified event just by increasing the influence of the positive instances in general. The nature of these changes is described by the equation 6.1. If those changes cause the (attribute, value) pair to be removed from the concept description, then the pair was not supposed to be

there in the first place. However, if the pair truly belongs to the concept description, the new positive instances will significantly reduce the contribution of the negative one, although never completely.

A related question to noise handling is one of enforcement and propagation of the constraints supplied externally to the classification process. The algorithm implemented in this work deals with the problem of constraints within the *process preparation* module. After inferring, where justified, the (attribute, value) pairs from the list of goal-relevant attributes in the descriptions of the events to be classified, the process-preparation module continues with the constraints-enforcement phase. Since the constraints are given in the form of (attribute, value) pairs as well, the system removes the forbidden pairs both from the event descriptions and the list of goal relevant attributes. Also, the (attribute, value) pairs covered by the pairs that represent the given constraints get removed from both structures. Since neither the event descriptions nor the list of goal-relevant attributes contain the forbidden pairs, the characterization and hierarchy-building processes (being a data-driven processes) will not introduce them into descriptions of the concepts at different levels of the hierarchy either. As a result, the (attribute, value) pairs that represent the constraints stated externally do not participate in the descriptions of the formed concepts and, consequently, cannot be propagated to the descriptions of the concepts at the higher levels of the hierarchy.

6.2.2.2. Retrieval

The efficient and flexible retrieval process is extremely important for human everyday behavior. Although the retrieval process itself is not of a major concern in this work, some of its properties that are directly influenced by the implemented approach to classification will be discussed briefly.

According to the definition of the similarity function given in Chapter 5, the similarity between the concept c and the event e , denoted as $link(c,e)$, is the function of their descriptions. It is fully defined by the equation 5.6. However, the class membership of the event cannot be determined on the basis of the descriptions only. The *link* parameter determines the degree of a match between the two descriptions and nothing else. In order to determine the class-membership of the event, the system needs several candidate concepts to choose from. What will decide which concepts are suitable candidates? The current context is the most plausible answer. The match between the context description and descriptions of the available concepts reduces the number of possible candidates to the ones that are relevant to the task at hand. Once the set of concepts has been decided on, the process of assigning class-membership can begin. The system computes the value of the *link* between the event and each of the concepts in the *candidate-set*. There is more than one strategy for assigning the event to a particular class (described by one of the concepts in the candidate-set) once the *link* computation is over. Two of them are discussed in section 5.4.5. of this dissertation. The fuzzy-set theory seems to be an appropriate approach to this problem as well.

However, whatever the chosen strategy of assigning an event to a particular class, it is important to notice that the whole process is a function of the following factors:

- the context of the task at hand,
- the set of existing concepts,
- the concept descriptions, and
- the event description.

Table 5.1 in Chapter 5 gives the values of the *link* between the command *ed* and classes described in figure 5.4, given the value of the relevance threshold in the concept descriptions is set to 0.5. It would be interesting to see the influence

of the change in the value of the threshold to the value of *link*. Table 6.1 extends table 5.1 in the sense that it introduces two more columns of the *link* values for the values of the relevance threshold of 0.26 and 0.7. The values of *link* are calculated again between the command *ed* and classes described in figure 5.4. The values in parentheses represent the number of (attribute, value) pairs in the class descriptions used as the basis of the calculation.

Table 6.1

class	<i>link</i>		
	threshold = 0.26	threshold = 0.5	threshold = 0.7
cat, stty	0.184 (19)	0.184 (19)	1.0 (2)
spell	0.273 (11)	0.273 (11)	0.273 (11)
nroff, style			
troff, troff-t	0.146 (12)	0.146 (12)	0.194 (9)
ed, ex, vi	0.303 (22)	0.516 (11)	0.667 (3)
lpr, print	0.188 (16)	0.188 (16)	0.25 (8)
spit-I	0.25 (12)	0.25 (12)	0.25 (12)
pr	0.25 (12)	0.25 (12)	0.25 (12)

There are several interesting aspects of table 6.1. First of all, the “correct” class (*ed, ex, vi*) maximizes the value of *link*. However, the implemented strategy of assigning class-membership would determine essentially the class to which the member belongs. Secondly, the lower the value of the threshold, the lower the value of *link*. Also, the lower the value of the threshold, the more specialized the concept description. Consequently, fewer events would match the concept description. Finally, a concept description *stabilizes* with more and more events being recognized as instances of the concept. Concepts with few instances are still

in the process of initial formation and may experience significant changes, much like humans after seeing just a couple of examples of a concept to be learned. The description of concepts they generate in those first moments is too general and covers lots of negative instances. That is the reason the *link* parameter has the value *1.0* between the class (*cat*, *stty*) and the event *ed*, given the threshold value of *0.7*. The concept description is too general to provide a plausible basis for the classification of a new instance. The same reason lies behind the constant values of *link* in the case of the singleton classes in table 6.1.

6.2.3. Efficiency Considerations

This topic has been discussed on several occasions in Chapter 5. Nevertheless, it is useful to emphasize positive (denoted by +) and negative (denoted by -) aspects of the efficiency of the implemented approach to classification.

- + There is neither backtracking nor extensive search. The clustering, characterization, and hierarchy-building processes proceed in the most plausible direction, as defined by the virtual and mental models, and never look back.
- + The main loop of the algorithm does not include the characterization process, unless a satisfying result of the clustering process has been found. Thus the characterization process does not participate in unfruitful attempts at classification.
- The process-preparation phase of the algorithm takes time to evaluate GDN rules.
- Maintaining and searching the auxiliary storage reduces the efficiency of the concept-description update process.
- Adding/removing the (attribute, value) pair to/from the concept description makes necessary the update of the relevance of other pairs in the concept description. However, this process should be a failure-driven one, for there is

no need to change anything unless the concept description fails to match a positive instance.

To be fair, it should be pointed out that the first (-) is the only real inefficiency related to the algorithm itself. The other two are concerned more with the update process. Also, the parallel-processing assumption, discussed earlier, would certainly change the whole situation drastically.

6.2.4. Semi-Automatic Generation of Event Representations

The algorithm, as defined in this chapter, requires the input events to be described in a pre-specified form. Any format, unless free, requires a certain effort on the part of a user. Because of that, the user may be reluctant to use the system in the first place. To avoid that, the feasibility of the semi-automatic generation of the required representation for the set of the input events has been tested.

The test was performed in the domain of the UNIX user-commands. Since UNIX maintains the on-line manual of all commands, the approach was to write a keyword look-up program that would, given a command, get the appropriate page of the manual, look for the set of the pre-defined keywords, and generate the corresponding (attribute, value) pairs once the appropriate keywords were found. The set of attributes and their values, used to describe the commands, played the role of the keywords. The program was written in Pascal (420 lines). The actual output of the program followed by the original description (coded manually) of the command is given below. The descriptions of two commands, *cat* and *ed*, are presented.

CAT

====

```
(is-a general-utility)
(attribute (function catenate-and-print 1.0))
(attribute (meaningful-mnemonic yes 1.0))
(attribute (domain file 1.0))
(attribute (type-of-parameters file 1.0))
(attribute (number-of-nonoptional-parameters 1 1.0))
(attribute (number-of-optional-parameters 100 1.0))
(attribute (number-of-flags 4 1.0))
(attribute (output-device standard-output 1.0))
(attribute (input-device standard-input 1.0))
(related-to (cp 1.0))
(related-to (ex 1.0))
(related-to (more 1.0))
(related-to (pr 1.0))
(related-to (tail 1.0))
```

```
(defschema cat
; "catenate and print"
  (attribute (function catenate-and-print 1.0))
  (is-a general-utility)
  (related-to (cp 1.0) (ex 1.0) (more 1.0) (pr 1.0) (tail 1.0))
  (attribute (domain user-file 1.0))
  (attribute (range file 1.0))
  (attribute (type-of-parameters user-file 1.0))
  (attribute (number-of-non-optional-parameters 1 1.0))
  (attribute (number-of-optional-parameters 100 1.0))
  (attribute (processing-time medium 1.0))
  (attribute (input-device standard-input 1.0))
  (attribute (output-device standard-output 1.0))
  (attribute (meaningful-mnemonic yes 1.0))
  (attribute (number-of-flags 4 1.0)))
```

ED

====

```
(is-a general-utility)
(attribute (function text-editor 1.0))
(attribute (meaningful-mnemonic yes 1.0))
(attribute (number-of-nonoptional-parameters 0 1.0))
(attribute (number-of-optional-parameters 1 1.0))
(attribute (number-of-flags 2 1.0))
(attribute (input-device standard-input 1.0))
(related-to (ex 1.0))
(attribute (output-device standard-output 1.0))
```

```

(defschema ed
;   "next editor"
  {attribute (function text-editing 1.0))
  {is-a general-utility}
  {related-to (ex 1.0))
  {attribute (domain user-file 1.0))
  {attribute (range user-file 1.0))
  {attribute (type-of-parameters user-file 1.0))
  {attribute (number-of-non-optional-parameters 0 1.0))
  {attribute (number-of-optional-parameters 1 1.0))
  {attribute (processing-time short 1.0))
  {attribute (input-device standard-input 1.0))
  {attribute (output-device disk 1.0))
  {attribute (meaningful-mnemonic yes 1.0))
  {attribute (number-of-flags 2 1.0)))

```

Although rather simple, the program has proved the point. It was able to create descriptions of the events from the input set. The quality of the generated descriptions varied from *poor* to *very good*, which was reasonable to expect since the goal of the experiment was to test the feasibility of the idea rather than implementing it to a full extent.

However, the program has demonstrated some other qualities as well. It never failed to come up with a correct *number-of-optional-parameters*, for instance, which happened to me several times while coding the command descriptions manually. Also, it never succeeded in returning a correct value for the attribute *processing-time*, simply because none of the command descriptions in the manual discuss that property at all. Then, obviously, there was no need for introducing it in the first place.

In general, the topic of the semi-automatic generation of the event representations is an important one, and deserves a serious research effort of its own. Any success in this area would make a significant contribution to the area of knowledge acquisition.

CHAPTER 7

IMPLEMENTATION

7.1. Domain of Application

The approach implemented in this work places the classification process in the broader context of the model of cognitive systems. The classification process interacts constantly with the other components of the model. It helps the problem-solving process to solve the task at hand. It helps the system as a whole to characterize the immediate environment and reduces the complexity of its description. The problem-solving process, in turn, serves as a performance system of the classification process. The expectations-generation and the spreading-activation processes supply the classification process with the additional information when needed. Most importantly, GDN rules serve as the driving force of the classification process itself.

Consequently, in order to test the performance of the classification component properly, the whole model of cognitive systems should be implemented. Obviously, this task requires much more extensive resources than could be offered by the work on a dissertation. However, limited testing of the feasibility of the approach can be a source of important information for future research as well. That was the reason behind the decision to implement the aspects of the classification algorithm that are independent of the rest of the system and assume the existence of the information normally supplied by the other components upon request from the classification process.

A subset of UNIX user commands is chosen as the domain of application of the classification algorithm. There are three reasons for that choice:

- UNIX is very popular at universities and research institutions. The result is that many people are bound to use it at some point in their careers.
- It is a complex and flexible system. It offers a variety of commands which can be combined in many possible ways.
- It is not an example of a user-friendly system. The mnemonic of its commands is not easy to remember and the system doesn't help you when the command is misspelled.

As a result, many people use a small percentage of this variety of cryptic commands. The idea is that the classification process may be employed to categorize the commands according to the degree of a match between a command and the properties defined by the goal and the context of the user's interaction with the system. Each category would contain a number of commands similar to each other with respect to the user needs. The user, then, chooses the command from the most appropriate category to solve his/her task at hand.

Since the ultimate goal is to implement the whole model and test the classification process in its intended environment, the implementation of the classification component is just the first step toward that goal. Having in mind the complexity of the model and variety of the types of problems involved, it is reasonable to assume that a powerful knowledge engineering tool, with a complex control and representational mechanisms, provides much better development environment than UNIX itself. Hence, ART (Inference Corporation, version 2.0), running on Texas Instruments' Explorer (Lisp machine), is chosen as the development environment for this application.

7.2. Generated Code

In order to implement the classification algorithm, the following program

segments have been generated:

- The *main* program - it includes process preparation, clustering, characterization, and the hierarchy-building modules. The complete listing of the program itself is given in Appendix A.
- The *update* program - it evaluates the resulting clustering and assigns either the blame or the credit to the corresponding (attribute, value) pairs in the list of the goal-relevant attributes according to equation 5.1. The update program, then, simulates the procedure of the actual update of the GDN rules that have added those pairs to the list. The final step is the second run of the clustering module. This run provides us with information on the effect of the modified list of goal-relevant attributes on the result of the clustering. A listing of the rules from the update program that differ from the ones found in the main program is given in Appendix B.
- The description of the *structure of attribute domains* - it is given in the form of rules. Each rule describes the relationship between two values of the attribute. A rule fires if there is the goal of inferring the value from its right-hand side, given that the left-hand side has matched the context description. A few sample rules are listed in Appendix C.

The list of actual attributes used to describe the UNIX user commands is given in Appendix D. The set of commands which provided samples for different tests of the algorithm is presented in Appendix E. Several examples of the actual representations of the commands have already been presented in Chapters 4 and 6.

7.3. Sample Runs

Let's assume that the user's goal is to write a letter to a sales representative of some computer manufacturer. Given below is the list of the (attribute, value,

relevance) triplets which are added to the list by the GDN rules that more or less match the description of the problem. Obviously, the size and the appropriateness of the list for the task at hand depends on the system's past experience with similar problems.

```
(defacts goal-rel-att-list
  (list goal-rel-att (function editing 1.0))
  (list goal-rel-att (function printing 0.9))
  (list goal-rel-att (function file-printing 0.9))
  (list goal-rel-att (function text-formatting 0.8))
  (list goal-rel-att (function find-spelling-errors 0.7))
  (list goal-rel-att (function sign-on 0.2))
  (list goal-rel-att (function set-terminal-options 0.05))
  (list goal-rel-att (domain user-file 1.0))
  (list goal-rel-att (range user-file 1.0))
  (list goal-rel-att (range printed-file 0.9))
  (list goal-rel-att (input-device standard-input 0.95))
  (list goal-rel-att (output-device line-printer 0.9))
  (list goal-rel-att (output-device laser-printer 0.6))
  (list goal-rel-att (number-of-non-optional-parameters 0 0.4)))
```

The list is used for classification of the following 14 commands (the input set is kept small to improve the readability of the output listing): *cat*, *ed*, *ex*, *lpr*, *nroff*, *pr*, *print*, *spell*, *spit-I*, *stty*, *style*, *troff*, *troff-t*, and *vi*. The listing of the complete run is given in Appendix F. Because of space limitations, some not-so-important details of the listing are not presented, which is emphasized by the italicized comments. Also, the awkward concept names, generated by the ART system, are replaced by more meaningful ones. The clustering phase of the same example is presented in figure 5.4.

If for some reason the user doesn't want to print the letter yet but rather to view it on the monitor, he/she may specify that in the form of a constraint:

```
(attribute-constraint (function printing))
```

The listing of the clustering phase for this case is given in Appendix G.

The value of the thresholds of the heuristic clustering-evaluation criteria is extremely important for the efficiency of the whole process and the quality of the final result. For example, if we change the threshold value of the distribution-of-events criterion from 0.5 to 0.65, the system produces 12 classes in 17 attempts, instead of 7 classes in 5 attempts described in Appendix F.

As pointed out in Chapter 5, the system learns from the past results of the clustering phase by updating the strength of the corresponding GDN rules. Figure 5.5 represents an example of the improvement of the clustering originally described in figure 5.4, after the process of the update has been employed. The listing of the actual run is presented in Appendix H. The rules implementing the updates are given in Appendix B.

Sometimes the list of goal-relevant (attribute, value) pairs generated by the GDN rules that match the goal and the current context of the classification does not provide sufficient information for a successful clustering. For example, if the user comes up with the problem of writing a piece of code in C language for his/her homework assignment, the list

```
(defacts goal-rel-att-list
  (list goal-rel-att (function editing 1.0))
  (list goal-rel-att (function C-program-compilation 1.0))
  (list goal-rel-att (function run-a-file 1.0))
  (list goal-rel-att (domain user-file 1.0))
  (list goal-rel-att (range user-file 1.0))
  (list goal-rel-att (input-device standard-input 0.95))
  (list goal-rel-att (output-device standard-output 0.8)))
```

is not sufficient for successful classification (with respect to the distribution-of-events criterion) of the following events: *cc*, *dbx*, *ed*, *ex*, *lint*, *lpr*, *pr*, *print*, and *vi*. After 9 attempts, the heuristic criterion is yet to be satisfied. However, upon consulting the expectations-generation process, the system may come up with a list that takes into consideration the same aspects of the previous, similar situations not found in this particular case.

```
(deffacts goal-rel-att-list
  (list goal-rel-att (function editing 1.0))
  (list goal-rel-att (function C-program-compilation 1.0))
  (list goal-rel-att (function run-a-file 1.0))
  (list goal-rel-att (function debugging 0.9))
  (list goal-rel-att (function printing 0.75))
  (list goal-rel-att (function C-program-verification 0.75))
  (list goal-rel-att (domain user-file 1.0))
  (list goal-rel-att (range user-file 1.0))
  (list goal-rel-att (range printed-file 1.0))
  (list goal-rel-att (input-device standard-input 0.95))
  (list goal-rel-att (output-device standard-output 0.8))
  (list goal-rel-att (output-device printer 0.75)))
```

The result of the clustering process (achieved after only 2 attempts) driven by the list given above is summarized in table 7.1.

Table 7.1

class #	commands
1	lint
2	lpr, pr, print
3	dbx
4	cc
5	ed, ex
6	vi

As pointed out earlier, there are some cases when neither the current context nor the common context alone offer a sufficient amount of information for a successful clustering. The associative links of the commands to be classified are employed in that case. Let's take the example of a novice user who wants to learn how to use UNIX. Where does he begin? Also, let's take the following 42 commands in the input set: *as, biff, cal, calendar, cmp, cp, date, dbx, du, file, find, finger, from, grep, iostat, last, leave, ln, mail, mail-user, msgs, mv, prmail,*

ps, *rm*, *sort*, *split*, *sysline*, *tail*, *talk*, *time*, *touch*, *uptime*, *users*, *vmstat*, *w*, *wall*, *which*, *who*, *write*, *xsend*, and *xget*. The list of goal-relevant (attribute, value) pairs

```
(deffacts goal-rel-att-list
  (list goal-rel-att (function file-manipulation 0.9))
  (list goal-rel-att (function mail-manipulation 0.8))
  (list goal-rel-att (function system-program 0.7))
  (list goal-rel-att (output-device printer 0.65))
  (list goal-rel-att (domain user-type-file 0.6))
  (list goal-rel-att (range control-info 0.6))
  (list goal-rel-att (range file-info 0.55)))
```

proves not to be complete enough to guide successfully the clustering process. The system, then, consults the *related-to* associative links of the commands from the input set. The *mail* command gets the support from 21% of the commands and earn the right to add the (attribute, value) pairs from its description to the list. The content of the list is given below.

```
(deffacts goal-rel-att-list
  (list goal-rel-att (function file-manipulation 0.9))
  (list goal-rel-att (function mail-manipulation 0.8))
  (list goal-rel-att (function system-program 0.7))
  (list goal-rel-att (output-device printer 0.65))
  (list goal-rel-att (domain user-type-file 0.6))
  (list goal-rel-att (range control-info 0.6))
  (list goal-rel-att (range file-info 0.55))
  (list goal-rel-att (function mail-reading 0.21))
  (list goal-rel-att (domain mailbox-file 0.21))
  (list goal-rel-att (range received-mail-message 0.21))
  (list goal-rel-att (number-of-nonoptional-parameters 0 0.21))
  (list goal-rel-att (number-of-optional-parameters 0 0.21))
  (list goal-rel-att (processing-time short 0.21))
  (list goal-rel-att (input-device standard-input 0.21))
  (list goal-rel-att (output-device standard-output 0.21))
  (list goal-rel-att (meaningful-mnemonic yes 0.21))
  (list goal-rel-att (number-of-flags 4 0.21)))
```

As a result, the system succeeds in clustering the set of input commands by generating 28 classes in 34 attempts. The number of generated classes is fairly high, which is caused by the system's attempt to produce the result of a classification

of high quality with respect to the distribution-of-events criterion. The system accepts the clustering if and only if the cardinality of each class is at least 0.5 of the average cardinality of all the generated classes, which is a pretty strong requirement. The lower the value of the threshold, the lower the number of the generated classes. Table 7.2 summarizes the number of attempts, the number of generated classes, the minimal acceptable cardinality, and the cardinality of the smallest generated class, for the above described example. It is obvious, then, that the number of generated classes would have been different had the value of the threshold been lower.

The main advantage of the associative links is that they are always able to supply the system with a few more (attribute, value) pairs, thus giving the clustering process another chance to generate the clustering which would satisfy the heuristic clustering-evaluation criterion.

Table 7.2

# of attempts	# of generated classes	minimal acceptable cardinality	minimal generated cardinality
1	4	6.825	2.0
2	5	5.460	2.0
3	6	4.550	2.0
4	7	3.900	2.0
5	8	3.413	2.0
7	9	3.033	2.0
8	11	2.482	1.0
9	12	2.275	1.0
10	13	2.100	1.0
12	14	1.950	1.0
13	16	1.706	1.0
14	17	1.606	1.0
15	18	1.517	1.0
20	19	1.437	1.0
21	20	1.365	1.0
22	21	1.300	1.0
25	22	1.241	1.0
26	23	1.187	1.0
27	24	1.138	1.0
28	25	1.092	1.0
32	26	1.050	1.0
33	27	1.011	1.0
34	28	0.975	1.0

7.4. Summary

The sample runs described in the preceding section point out the importance of GDN, the virtual model, and the mental model for the plausibility of the result of classification. GDN guides the clustering process. The virtual model contains a description of the goal and the context of the task at hand. The mental model provides a natural environment for both of them. The quality of their description, capturing past experience, determines the quality of the classification. The classification process itself is rather simple. It is the interaction with other components of the cognitive system that makes it both complex and powerful at the same time.

CHAPTER 8

CONCLUSION

8.1. Contributions of the Dissertation

There are several aspects of this work that are worth emphasizing:

- *The main goal of the work is to outline the system capable of classifying the set of given events so that the resulting classification is useful for the solution of the problem at hand.* It defines the mechanisms that provide the system with the ability to adapt easily and efficiently to novel situations.
- *The implemented approach places the classification process within the cognitive system as a whole.* It emphasizes the viewpoint that if the system is to match the human classification performance, it cannot consider classification in isolation from other aspects of human behavior. Only the full interaction among all the components of the cognitive system gives meaning to the performance of each of them. Consequently, the performance of the classification component of a cognitive system depends on the intensity and the quality of the interactions with the rest of the system.
- *The mental and virtual models relate the cognitive system to the environment surrounding it.* The mental model reflects past experience of the system. It consists of all the knowledge the system has gathered through the day-to-day interactions with the external world. After some changes in the immediate environment have caught its attention, the system creates a description of the environment and allocates the appropriate resources to ensure the most plausible response to the challenge from the world. The description of the environment and the allocated resources form a virtual model. The

virtual model, then, provides the interface between the outside world and the mental model. It provides a reference to the specificities of the current task. As a result, the classification component receives, through the virtual model, not only a description of the events to be classified, but also the goal and the context of the classification as well. Even more, the virtual model contains background knowledge (provided by the mental model) in the form of the structure of the attribute domains and the GDN rules that match the goal and the context of the classification. The information supplied by the virtual model, then, increases the degree of usefulness of the result of the classification with respect to the task at hand.

- *Classification is viewed as both the data- and model-driven process.* The clustering phase of the process is efficiently guided by the model embedded in the form of the list of goal-relevant attributes, which is, in turn, generated by the virtual model. In the process of generating the list of goal-relevant attributes, the following sources of information have been consulted: the goal of the classification, the context of the task at hand, the context of similar situations experienced in the past, and the associative links of the events from the input set. Which sources of information are consulted in any particular case, and in which order, is determined according to the visibility parameter.

The characterization and the hierarchy-building phases of the classification process are data-driven. Contrary to the clustering process, which employs the model to reduce the amount of information to be considered during the process, the characterization and the hierarchy-building processes use as much information as possible from the descriptions of the input events in order to make the descriptions of the generated classes more complete and flexible for future retrievals.

The implemented mechanism for calculating the relevance of the (attribute, value) pairs used in concept descriptions and a similar mechanism for computing the strength of the hierarchical links further ensure the flexibility of the generated concepts with respect to their future use.

- *The approach described in this work emphasizes the use of heuristic rules at different levels of the classification process.* The three phases of the clustering process are ordered according to the visibility parameter which is of a heuristic nature.

The resulting clustering is evaluated by the heuristic criteria.

The update process is heuristically driven as well. All the update equations defined in Chapter 5 represent yet another heuristic.

The values of different thresholds introduced throughout the study are defined in a heuristic manner.

- *Two kinds of feedback are defined in this work: an internal and an external one.* The problem-solving process is recognized as the performance system of the classification process, thus providing the internal feedback. The definitions of both the model of cognitive systems and the mental model assume that the result of classification must be evaluated in the context of the process that requested it in the first place in its attempt to solve a particular problem.

The external feedback is provided by the external user when the clustering process fails to produce a successful clustering of the set of given events.

- *There are two levels of updates performed by the system.* Upon receiving the feedback from the problem-solving process, the system modifies the value of the heuristic clustering-evaluation criteria. The heuristic criteria, in turn, provide the feedback for the sources of information used in the process of

clustering: GDN rules, GDN threshold, and the associative links of the input events. The classification algorithm, which incorporates all the specificities of the approach described in this work, is given. The algorithm is tested in the domain of the UNIX user commands and points out the importance of the information supplied by the virtual model for the success of the classification.

- *The work described here doesn't stop with the concepts generated through the process of classification. It outlines the mechanism for incremental learning embedded in the process of updating the description of the concept for which a new instance has been encountered.* The same mechanism makes the system robust with respect to the noise in the generated clustering of the set of input events.
- *The aspects of the retrieval process are discussed as well.* The similarity function is defined in accordance with the approach outlined here and the adopted knowledge representation mechanism. The outcome of the retrieval process, then, depends on the similarity function, the concepts (and their descriptions) already known to the system, the context of the task at hand, and the description of the event to be assigned to a particular class.

8.2. Future Research Areas

There are several directions in which this work can be extended:

- *Development of the components of a cognitive system which influence directly the performance of the classification component* - once the whole system is put together, it can be tested against human performance on the same problem.
- *The process of creating GDN* - since the quality of the rules from GDN determines the quality of the resulting classification, special attention must

be paid to the process of their generation. Several components participate in that process. They should be isolated and described. Many questions related to the consistency, appropriateness, and flexibility of the rules will have to be answered.

- *Defining the additional heuristics for clustering-evaluation criterion* - the more heuristics, the better the criterion and, consequently, the better the quality of the resulting clustering. Also, the additional heuristic would improve the system's ability to react properly to a greater variety of different situations, thus improving its generality and applicability to different application domains.
- *Evaluating and tuning the performance of the update mechanism in the long run* - it is only after the system is fully developed and tested on a large set of examples that the update mechanism may be effectively evaluated. Its limitations must be carefully analyzed and removed.
- *Testing the classification algorithm in new application domains* (e.g. database design) - portability to new domains is an important property of a classification algorithm. The specificities of the different domains would help the process of tuning the algorithm as a whole.
- *Improving the process of the semi-automatic generation of the event descriptions* - the work initiated here has proved only the feasibility of the approach. The actual mechanism that would generate the descriptions of the input events, thus freeing the user from that burden, is yet to be developed.
- *Classification of the events described in terms of their components and the relationships among them* - it is necessary to suggest both the appropriate representation of such events and the corresponding changes in the classification algorithm that would extend the domain of applicability of the implemented approach to structured events as well.

This extensive list of possible areas of future research emphasizes the importance of the topic discussed in this study.

APPENDIX A

THE MAIN PROGRAM

```

;;; -*- Mode: ART; Base: 10; Package: ART-USER -*-
;;; This file contains the rules that implement
;;; the clustering and characterization phases
;;; of the classification algorithm.

... *****
;;; *
;;; *
;;; *      Definitions of Relations
;;; *
;;; *
... *****
;;;

(defrelation threshold
  (?threshold-name ?value))

(defrelation attribute-constraint
  (?attribute ?value))

(defrelation concept-attribute
  (?schema (?attribute ?value ?relevance)))

(defrelation number-of-concepts-constraint
  (?value))

(defrelation number-of-concepts
  (?value))

(defrelation infer
  (?schema (?attribute ?value ?relevance)))

(defrelation flag-triplet
  (?schema (?attribute ?value ?relevance)))

(defrelation new-list
  (?name (?attribute ?value ?relevance)))

(defrelation temp-list
  (?name (?attribute ?value ?relevance)))

(defrelation temp-list-2
  (?name ?attribute))

(defrelation list-length
  (?list ?length))

```

```

(defrelation list-length-2
  (?list ?length))

(defrelation top-relevance
  (?list ?att ?val))

(defrelation concept
  (?att ?val))

(defrelation concept-card
  (?concept ?cardinality))

(defrelation concept-max-card
  (?concept ?cardinality))

(defrelation concept-length
  (?concept ?length))

(defrelation temp-concept-length
  (?concept ?length))

(defrelation print-concept
  (?concept))

(defrelation temp-concept-6
  (?concept))

(defrelation temp-concept-card
  (?concept ?cardinality))

(defrelation average-concept-cardinality
  (?value))

(defrelation concept-att-val
  (?concept ?attribute ?value ?total-relevance))

(defrelation concept-att-val-copy
  (?attribute ?concept ?event ?value))

(defrelation concept-att-card
  (?concept ?event ?attribute-cardinality))

(defrelation print-concept-description
  (?concept))

(defrelation member-of
  (?concept ?event))

(defrelation temporary-1-member-of
  (?concept ?event))

(defrelation temporary-2-member-of
  (?concept ?event))

```

```
(defrelation new-member-of
  (?concept ?event))

(defrelation temp-member-info
  (?concept ?event))

(defrelation temp-member-info-2
  (?concept ?event))

(defrelation go-print1
  (?concept))

(defrelation go-print2
  (?concept))

(defrelation partic-att-val
  (?concept ?event ?attribute ?value ?multiplied-rel))

(defrelation instance-link-sum-strength
  (?concept ?event ?strength-sum))

(defrelation number-of-given-events
  (?number))

(defrelation att-val-cp
  (?attribute ?event ?value ?relevance))

(defrelation successful-clustering
  (?yes-no))

(defrelation heuristic-rule-strength
  (?rule-name ?strength))

(defrelation valid-criterion
  (?name))

(defrelation passed-criterion
  (?name))

(defrelation do-it-again
  (?yes-no))

(defrelation trigger-0
  (?on-off))

(defrelation trigger-1
  (?on-off))

(defrelation trigger-2
  (?on-off))

(defrelation trigger-3
  (?on-off))
```

```
(defrelation trigger-4
  (?on-off))

(defrelation trigger-5
  (?on-off))

(defrelation trigger-6
  (?on-off))

(defrelation trigger-7
  (?on-off))

(defrelation trigger-8
  (?on-off))

(defrelation trigger-9
  (?on-off))

(defrelation trigger-10
  (?on-off))

(defrelation trigger-11
  (?on-off))

(defrelation trigger-12
  (?on-off))

(defrelation trigger-13
  (?on-off))

(defrelation trigger-14
  (?on-off))

(defrelation trigger-15
  (?on-off))

(defrelation trigger-17
  (?on-off))

(defrelation instance
  (?concept (?event ?relevance)))

(defrelation instance-of-concept
  (?event (?concept ?relevance)))

(defrelation subordinate-concept
  (?concept (?sub-concept ?relevance)))

(defrelation superordinate-concept
  (?concept (?sup-concept ?relevance)))

(defrelation count-level
  (?level))
```

```
(defrelation attempt
  (?attempt-number))
```

```
... *****
??? *
... *
??? *
... *
??? *
... *****
??? *
```

Definition of Facts

```
(deffacts thresholds
  (threshold goal-att-strength 0.1)
  (threshold heuristics 0.5)
  (threshold distribution-of-events-criterion 0.5)
  (threshold concept-att 0.5)
  (threshold instance-link 0.1)
  (threshold instance-of-link 0.1)
  (threshold subordinate-link 0.1)
  (threshold super-ordinate-link 0.1))
```

```
(deffacts initial-state
  (top-relevance goal-rel-att function 0.0)
  (successful-clustering no)
  (heuristic-rule-strength number-of-concepts 1.0)
  (heuristic-rule-strength distribution-of-events 1.0)
  (number-of-concepts 0.0)
  (number-of-given-events 0.0)
  (attempt 0)
  (concept-length ((universe all)) 0)
  (count-level 0)
  (concept-max-card ((universe all)) 0)
  (list-length goal-rel-att 0)
  (list-length-2 goal-rel-att 0)
  (do-it-again no)
  (trigger-1 on)
  (trigger-2 on)
  (trigger-3 on)
  (trigger-5 on)
  (trigger-6 on)
  (trigger-13 on)
  (trigger-15 on))
```

```
... *****
??? *
... *
??? *
... *
??? *
... *****
??? *****
??? *****
??? *****
```

Classification Process

```
... *****
??? *
... *
??? *
... *
??? *
```

Initial State

```

... *****
;;;

... *
;;; * Next Iteration Preparation
;;; *****
... *
;;;

(defrule initialize-next-iteration-1
  (declare (salience 60))
  (trigger-4 on)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  ?y <- (list new-list (?att ?val ?rel))
=>
  (retract ?x ?y)
  (assert
    (list goal-rel-att (?att ?val ?rel))))

(defrule initialize-next-iteration-2
  (declare (salience 60))
  (trigger-4 on)
  ?x <- (member-of ?concept ?schema)
  ?y <- (new-member-of ?concept ?schema)
=>
  (retract ?x ?y)
  (assert
    (member-of ?concept ?schema)))

(defrule garbage-collection-2
  (declare (salience 60))
  (trigger-4 on)
  ?x <- (passed-criterion ?val)
=>
  (retract ?x))

(defrule remove-trigger4
  (declare (salience 50))
  ?x <- (trigger-4 on)
=>
  (retract ?x))

... *
;;; * Clustering Message
;;; *****
... *
;;;

(defrule print-clustering-message
  (declare (salience 45))
  (trigger-15 on)
=>
  (printout t t t t t "CLUSTERING PROCESS" t "*****"))

... *
;;;

```

```

;;; * Calculate the Number of Given Events
;;; *****
;;; *
;;;

(defrule event-set-generation
  (declare (salience 10))
  (trigger-15 on)
  (is-a ?event ?sup-concept)
  (attribute ?event (function ?val ?rel))
=>
  (assert
    (member-of ((universe all)) ?event)
    (temp-member-info ((universe all)) ?event)))

(defrule determine-number-of-given-events
  (declare (salience 9))
  (trigger-15 on)
  ?x <- (temp-member-info ((universe all)) ?schema)
  ?y <- (number-of-given-events ?num)
=>
  (retract ?x ?y)
  (assert
    (number-of-given-events =(?num + 1))))

(defrule print-number-of-events
  (declare (salience 8))
  (trigger-15 on)
  (number-of-given-events ?val)
=>
  (printout t t t t "there are " ?val " events in the initial set"))

(defrule initial-concept-cardinality
  (declare (salience 7))
  (trigger-15 on)
  (number-of-given-events ?val)
=>
  (assert
    (concept-card ((universe all)) ?val)))

(defrule remove-trigger-15
  (declare (salience 6))
  ?x <- (trigger-15 on)
=>
  (retract ?x))

;;; *
;;; * Modify Event Descriptions
;;; *****
;;; *
;;;

(defrule mark-the-triplets
  (declare (salience -6))
  (trigger-1 on)

```

```

(OR
  (list ?schema (?att ?val ?rel))
  (attribute ?schema (?att ?val ?rel)))
=>
  (assert
    (infer ?schema (?att ?val ?rel))))

(defrule modify-event-description
  (declare (salience -7))
  (trigger-1 on)
  (list goal-rel-att (?att ?val ?rel))
  (member-of ((universe all)) ?event)
  (not (attribute ?event (?att ?val ?rel1)))
  (infer ?event (?att ?val ?rel1))
=>
  (assert
    (attribute ?event (?att ?val ?rel1)))
  (printout t t t "inferred (" ?att " " ?val " " ?rel1 ") triplet"
    t " in the description of " ?event))

... *
... * Take Into Account Given Constraints
... *****
... *
... *

(defrule constraint-remove-1a
  (declare (salience -10))
  (trigger-1 on)
  (attribute-constraint ?att ?val)
  (list goal-rel-att (?att ?val ?rel))
  (infer goal-rel-att (?att ?val ?rel1))
=>
  (printout t t t t "remove from the goal-rel-att list the (att, val) pairs"
    t " covered by (" ?att " " ?val " ): given constraint"))

(defrule constraint-remove-1b
  (declare (salience -11))
  (trigger-1 on)
  (attribute-constraint ?att ?val)
  (member-of ((universe all)) ?event)
  (attribute ?event (?att ?val ?rel))
  (infer ?event (?att ?val ?rel1))
=>
  (printout t t t t "remove from the event " ?event " description
    the (att, val) pairs" t " covered by (" ?att " " ?val " ):
    given constraint"))

... *
... * Climb a Domain Hierarchy
... * (Goal-Rel-Att List)
... *****
... *
... *

```

```

(defrule climbing-domain-hierarchy-1
  (declare (salience -12))
  (trigger-1 on)
  (list goal-rel-att (?att ?val ?rel))
  (infer goal-rel-att (?att ?val ?rel1))
=>
  (printout t t t t "remove from the goal-rel-att list the (att, val) pairs"
    t " covered by (" ?att " " ?val "):" t " climbing a domain hierarchy"))

;;; *
;;; * Remove Marked Triplets
;;; *****
;;; *
;;; *

(defrule constraint-remove-1c
  (declare (salience -13))
  (trigger-1 on)
  (attribute-constraint ?att ?val)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  ?y <- (infer goal-rel-att (?att ?val ?rel))
=>
  (retract ?x ?y)
  (printout t t t t "removed (" ?att " " ?val " " ?rel ") from the goal-rel-att list:"
    t " given constraint"))

(defrule remove-marked-list-triplets
  (declare (salience -14))
  (trigger-1 on)
  ?x <- (flag-triplet goal-rel-att (?att ?val ?rel))
  ?y <- (list goal-rel-att (?att ?val ?rel))
=>
  (retract ?x ?y)
  (printout t t t t "removed (" ?att " " ?val " " ?rel ") from the goal-rel-att list:"
    t " already covered"))

(defrule constraint-remove-1d
  (declare (salience -15))
  (trigger-1 on)
  (attribute-constraint ?att ?val)
  (member-of ((universe all)) ?event)
  ?x <- (attribute ?event (?att ?val ?rel))
  (infer ?event (?att ?val ?rel))
=>
  (retract ?x)
  (printout t t t t "removed (" ?att " " ?val " " ?rel ") from the event " ?event
    " description:" t " given constraint"))

(defrule remove-marked-triplets
  (declare (salience -16))
  (trigger-1 on)
  (attribute-constraint ?att ?val1)
  (infer ?schema (?att ?val1 ?rel1))
  ?x <- (flag-triplet ?schema&~goal-rel-att (?att ?val ?rel))

```

```

=> ?y <- (attribute ?schema (?att ?val ?rel))
(retract ?x ?y)
(printout t t t t "removed (" ?att " " ?val " " ?rel ") from the description of "
?schema ":" t " already covered"))

;;; *
;;; * Remove the Att-Val Pairs With a Low Relevance
;;; * (Goal-Related Attributes List)
;;; *****
;;; *
;;; *

(defrule threshold-remove-1
  (declare (salience -17))
  (trigger-1 on)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  (threshold goal-att-strength ?th-val)
  (test
    (<= ?rel ?th-val))
=>
  (retract ?x)
  (printout t t t t "removed (" ?att " " ?val " " ?rel ") from the goal-rel-att list:"
    t " low relevance"))

;;; *
;;; * Remove the Att-Val Pairs Not Covering Any Event
;;; *****
;;; *
;;; *

(defrule remove-not-used-attribute-value-pair
  (declare (salience -20))
  (trigger-1 on)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  (trigger-1 on)
  (forall
    (member-of ((universe all)) ?event)
    (not (attribute ?event (?att ?val ?rel))))
=>
  (retract ?x)
  (printout t t t t "removed (" ?att " " ?val " " ?rel
    ") triplet from the goal-rel-att list:" t " doesn't cover any event"))

;;; *
;;; * Calculate the Number of Goal-Related Attributes
;;; *****
;;; *
;;; *

(defrule list-length-initialization
  (declare (salience -23))
  (trigger-1 on)
  (list goal-rel-att (?att ?val ?rel))

```

```

=>
  (assert
    (temp-list goal-rel-att (?att ?val ?rel))
    (temp-list-2 goal-rel-att ?att)))

(defrule determine-number-of-goal-rel-atts
  (declare (salience -24))
  (trigger-1 on)
  ?x <- (temp-list goal-rel-att (?att ?val ?rel))
  ?y <- (list-length goal-rel-att ?length)
=>
  (retract ?x ?y)
  (assert
    (list-length goal-rel-att =(?length + 1))))

(defrule print-list-length
  (declare (salience -25))
  (trigger-1 on)
=> (declare (salience -25))
  (trigger-1 on)
  (list-length goal-rel-att ?length)
  (list-length goal-rel-att t "number of generated goal-related (att, val) pairs = " ?length))

(defrule determine-different-goal-rel-atts
  (declare (salience -26))
  (trigger-1 on)
  (list-length-2 goal-rel-att ?length)
  ?x <- (temp-list-2 goal-rel-att ?att)
=>
  (retract ?x)
  (assert
    (list-length-2 goal-rel-att =(?length + 1))))

... *
... * Store the List
... *****
... *
... *

(defrule save-list
  (declare (salience -27))
  (trigger-1 on)
  (list goal-rel-att (?att ?val ?rel))
=>
  (assert
    (list goal-att-copy (?att ?val ?rel))))

... *
... * Administration
... *****
... *
... *

(defrule remove-trigger
  (declare (salience -28))

```

```

=>    ?x <- (trigger-1 on)
      (retract ?x))

(defrule remove-infer-messages
  (declare (salience -30))
  ?x <- (infer ?schema (?att ?val ?rel))
=>    (retract ?x))

... *****
... *
... *
... *           Clustering Process
... *
... *****
...

... *
... * Current Level of the Hierarchy
... *****
... *
...

(defrule determine-concept-with-max-cardinality
  (declare (salience -31))
  (successful-clustering no)
  (trigger-13 on)
  (concept-length ?concept ?length)
  (list-length-2 goal-rel-att ?length1)
  (test
    (< ?length ?length1))
  (concept-card ?concept ?card)
  ?x <- (concept-max-card ?concept1 ?card1)
  (test
    (> ?card ?card1))
=>    (retract ?x)
      (assert
        (concept-max-card ?concept ?card)))

(defrule suspend-when-max-card-1
  (declare (salience -33))
  ?x <- (concept-max-card ?concept ?card)
  (test
    (<= ?card 1))
=>    (retract ?x))

(defrule garbage-collection-1
  (declare (salience -35))
  (successful-clustering no)
  (trigger-13 on)
  ?x <- (concept-card ?concept ?card)
=>    (retract ?x))

```

```

(defrule count-level-of-hierarchy-1
  (declare (salience -37))
  (successful-clustering no)
  (trigger-13 on)
  (concept-max-card ?concept ?card)
  (test
    (> ?card 1))
  (concept-length ?concept ?length)
=>
  (printout t t t t "LEVEL " ?length))

(defrule attempt-1
  (declare (salience -39))
  ?x <- (trigger-13 on)
  ?y <- (attempt ?attempt)
=>
  (retract ?x ?y)
  (assert
    (attempt =(?attempt + 1))))

(defrule attempt-2
  (declare (salience -40))
  (attempt ?attempt)
  (concept-max-card ?concept ?card)
  (test
    (> ?card 1))
=>
  (printout t " - ATTEMPT " ?attempt))

;;; *
;;; * Get the First Applicable Attribute From
;;; * the List of Goal-Related Attributes
;;; *****
;;; *****
;;; *
;;; *
(defrule get-attribute
  (declare (salience -42))
  (successful-clustering no)
  (list goal-rel-att (?att1 ?val1 ?rel1))
  (not
    (exists
      (concept-max-card ($? (?att1 ?val2) $?) ?card)))
  ?x <- (top-relevance goal-rel-att ?att2 ?rel2)
  (test
    (> ?rel1 ?rel2))
=>
  (retract ?x)
  (assert
    (top-relevance goal-rel-att ?att1 ?rel1)))

(defrule defining-attribute-message
  (declare (salience -43))

```

```

    (successful-clustering no)
    (top-relevance goal-rel-att ?att ?rei)
    (concept-max-card $?concept ?card)
    (test
      (> ?card 1))
=>
  (printout t t t t "concept: " (list$ ?concept) " - classifying attribute: " ?att))

```

```

;;; *
;;; * Generate a Concept for Each Value of the Attribute
;;; *****
;;; *
;;; * Determine Instances of a Concept
;;; *
;;; * (A concept with no instances will not
;;; *   be evaluated in the next iteration.)
;;; *****
;;; *****
;;; *

```

```

(defrule membership-determination
  (declare (salience -44))
  (successful-clustering no)
  (concept-max-card ($?concept) ?card)
  (concept ?att ?val)
  ?x <- (member-of ($?concept) ?event)
  (attribute ?event (?att ?val ?rel))
=>
  (retract ?x)
  (assert
    (member-of ($?concept (?att ?val)) ?event)
    (temporary-1-member-of ($?concept (?att ?val)) ?event)))

```

```

(defrule other-value-concepts
  (declare (salience -47))
  (successful-clustering no)
  (concept-max-card ($?concept) ?card)
  (concept ?att ?val)
  ?x <- (member-of ($?concept) ?event)
=>
  (retract ?x)
  (assert
    (member-of ($?concept (?att other)) ?event)
    (temporary-2-member-of ($?concept (?att other)) ?event)))

```

```

(defrule update-the-length-of-concept
  (declare (salience -48))
  (successful-clustering no)
  (concept-max-card ($?concept1) ?card)
  (member-of ($?concept1 ?concept2) ?event)
  (not (concept-length ($?concept1 ?concept2) ?length1))
  (concept-length ($?concept1) ?length)
=>

```

```

(assert
  (concept-length ($?concept1 ?concept2) =(length + 1))))

;;; *
;;; * Print Concept Instances
;;; *****
;;; *

(defrule concepts-generator
  (declare (salience -45))
  (concept-max-card ?concept ?card)
  (test
    (> ?card 1))
  (top-relevance goal-rel-att ?att ?rel)
  (list goal-rel-att (?att ?val ?rel1))
=>
  (assert
    (concept ?att ?val))
  (printout t t t "following concept is defined by the attribute " ?att
    t " and value " ?val ":"))

(defrule print-concept-instances-1
  (declare (salience -44))
  (successful-clustering no)
  (member-of $?concept ?event)
  ?x <- (temporary-1-member-of $?concept ?event)
=>
  (retract ?x)
  (printout t t (list$ ?concept) " with an instance " ?event))

(defrule print-a-heading
  (declare (salience -46))
  (concept-max-card ?concept ?card)
  (test
    (> ?card 1))
  (top-relevance goal-rel-att ?att ?rel)
=>
  (printout t t t "following concept is defined by the attribute " ?att
    t " and value OTHER:"))

(defrule print-concept-instances-2
  (declare (salience -47))
  (successful-clustering no)
  (member-of $?concept ?event)
  ?x <- (temporary-2-member-of $?concept ?event)
=>
  (retract ?x)
  (printout t t (list$ ?concept) " with an instance " ?event))

;;; *
;;; * Garbage Collection
;;; *****

```

```

;;; *
(defrule remove-concept-messages
  (declare (salience -48))
  (successful-clustering no)
  ?x <- (concept ?att ?val)
=>
  (retract ?x))

;;; *
;;; * Apply the Heuristic Criteria
;;; *****
;;; *****
;;; *
;;; *

;;; *
;;; * Preparatory Calculation
;;; *****
;;; *
;;; *

(defrule wake-up-rule
  (declare (salience -48))
  (successful-clustering no)
  (member-of ?concept ?event)
  (not (trigger-17 on))
=>
  (assert
    (trigger-17 on)))

(defrule member-info-copy
  (declare (salience -49))
  (successful-clustering no)
  (trigger-17 on)
  (member-of ?concept ?event)
=>
  (assert
    (temp-member-info ?concept ?event)))

(defrule concept-info
  (declare (salience -50))
  (successful-clustering no)
  (trigger-17 on)
  (member-of ?concept ?event)
  (not (concept-card ?concept ?val))
=>
  (assert
    (concept-card ?concept 0.0)))

(defrule concept-cardinality
  (declare (salience -51))
  (successful-clustering no)
  ?x <- (concept-card ?concept ?val)
  ?y <- (temp-member-info ?concept ?event)

```

```

=>
  (retract ?x ?y)
  (assert
    (concept-card ?concept =(?val + 1))))

(defrule copy-concept-info
  (declare (salience -52))
  (successful-clustering no)
  (concept-card ?concept ?val)
=>
  (assert
    (temp-concept-card ?concept ?val)))

(defrule count-concepts
  (declare (salience -53))
  (successful-clustering no)
  ?x <- (temp-concept-card ?concept ?card)
  ?y <- (number-of-concepts ?val)
=>
  (retract ?x ?y)
  (assert
    (number-of-concepts =(?val + 1))))

;;; *
;;; * Print the Result of the Clustering Process
;;; *****
;;; *****
;;; *
;;;

(defrule print-number-of-concepts
  (declare (salience -54))
  (successful-clustering no)
  (number-of-concepts ?number)
  (concept-max-card ?concept ?card)
  (test
    (> ?card 1))
=>
  (printout t t t t "there are " ?number " concepts"))

(defrule print-cardinality-of-concepts
  (declare (salience -55))
  (successful-clustering no)
  (concept-card $?concept ?card)
  (concept-max-card ?concept1 ?card1)
  (test
    (> ?card1 1))
=>
  (printout t t "cardinality of the concept " (list$ ?concept) " is " ?card))

;;; *
;;; * Define a Heuristic Criterion to be Applied
;;; * (Strength Above the Threshold)

```

```

... *****
;;;
;;; *
;;;

(defrule define-criteria-1
  (declare (salience -56))
  (successful-clustering no)
  (heuristic-rule-strength ?val&~number-of-concepts ?strength1)
  (not (valid-criterion ?val))
  (threshold heuristics ?strength2)
  (test
    (> ?strength1 ?strength2))
=>
  (assert
    (valid-criterion ?val))
  (printout t t t t "heuristic criterion " ?val " is going to be applied"))

(defrule define-criteria-2
  (declare (salience -56))
  (successful-clustering no)
  (number-of-concepts-constraint ?value)
  (heuristic-rule-strength number-of-concepts ?strength1)
  (not (valid-criterion number-of-concepts))
  (threshold heuristics ?strength2)
  (test
    (> ?strength1 ?strength2))
=>
  (assert
    (valid-criterion number-of-concepts))
  (printout t t t t "heuristic criterion NUMBER-OF-CONCEPTS is going to be applied"))

;;; *
;;; * Heuristic Criterion 1:
;;; *   Predefined Number of Concepts To Be Generated
;;; *****
;;;
;;; *
;;;

(defrule number-of-concepts-heuristic-criterion-1
  (declare (salience -57))
  (successful-clustering no)
  (valid-criterion number-of-concepts)
  (number-of-concepts-constraint ?val1)
  (number-of-concepts ?val2)
  (test
    (= ?val1 ?val2))
=>
  (assert
    (passed-criterion number-of-concepts))
  (printout t t t t "heuristic criterion NUMBER-OF-CONCEPTS is satisfied"))

(defrule number-of-concepts-heuristic-criterion-2
  (declare (salience -57))
  (successful-clustering no)
  (valid-criterion number-of-concepts)

```

```

(number-of-concepts-constraint ?val1)
(number-of-concepts ?val2)
(test
 (< ?val1 ?val2))
=>
(assert
 (passed-criterion number-of-concepts))
(printout t t t t "heuristic criterion NUMBER-OF-CONCEPTS is not satisfied, but"
 t " the number of concepts is greater than required"
 t " and can be reduced to predefined number"))

... *
... *
... * Heuristic Criterion 2:
... *
... * Roughly Even distribution of the Events
... * Across the Generated Concepts
... *
... *
... *
... *
... *

(defrule distribution-of-events-heuristic-criterion-1
  (declare (salience -58))
  (successful-clustering no)
  (concept-max-card ?concept5 ?card5)
  (test
   (> ?card5 1))
  (valid-criterion distribution-of-events)
  (number-of-given-events ?total)
  (number-of-concepts ?concept-no)
  (threshold distribution-of-events-criterion ?t)
=>
  (assert
   (average-concept-cardinality =((?total / ?concept-no) * ?t))))

(defrule print-cardinality
  (declare (salience -59))
  ?x <- (trigger-6 on)
  (average-concept-cardinality ?val)
  (concept-max-card ?concept5 ?card5)
  (test
   (> ?card5 1))
=>
  (retract ?x)
  (printout t t t t "minimal cardinality = " ?val))

(defrule distribution-of-events-heuristic-criterion-2
  (declare (salience -62))
  ?x <- (trigger-2 on)
  ?y <- (average-concept-cardinality ?con-card)
  (forall
   (concept-card ?concept ?card)
   (test
    (>= ?card ?con-card)))
=>
  (retract ?x ?y)

```

```

(assert
  (passed-criterion distribution-of-events))
(printout t t t "heuristic criterion DISTRIBUTION-OF-EVENTS is satisfied"))

... *
... * Evaluate the Heuristic Criteria
... *****
... *
... *

(defrule stopping-criterion
  (declare (salience -64))
  ?x <- (trigger-3 on)
  ?y <- (successful-clustering no)
  (forall
    (valid-criterion ?val)
    (passed-criterion ?val))
=>
  (retract ?x ?y)
  (assert
    (successful-clustering yes))
  (printout t t t "clustering process is COMPLETED SUCCESSFULLY"))

... *
... * Initialization (If Not Successful)
... *****
... *****
... *
... *

(defrule initialization-1
  (declare (salience -66))
  (successful-clustering no)
  (list goal-rel-att (?att ?val ?rel))
=>
  (assert
    (list new-list (?att ?val ?rel))))

(defrule initialization-2
  (declare (salience -67))
  (successful-clustering no)
  (member-of ?concept ?schema)
=>
  (assert
    (new-member-of ?concept ?schema)))

(defrule remove-average-cardinality-message
  (declare (salience -68))
  (successful-clustering no)
  ?x <- (average-concept-cardinality ?con-card)
=>
  (retract ?x))

(defrule initialization-10

```

```

(declare (salience -69))
(successful-clustering no)
?x <- (number-of-concepts ?number&~0.0)
?y <- (trigger-17 on)
?z <- (top-relevance goal-rel-att ?att ?rel&~0.0)
?w <- (concept-max-card ?concept ?card&~0.0)
(test
  (> ?card 1))
=>
(retract ?x ?y ?z ?w)
(assert
  (number-of-concepts 0.0)
  (top-relevance goal-rel-att ?att 0.0)
  (concept-max-card ?concept 0.0)
  (trigger-2 on)
  (trigger-4 on)
  (trigger-6 on)
  (trigger-13 on)
  (trigger-17 on))
(printout t t t "UNSUCCESSFUL clustering => initialization process"))

;;; *
;;; * Print the Result of the Clustering Process
;;; *****
;;; *
;;;

(defrule print-clustering-results-1
  (declare (salience -75))
  (trigger-5 on)
  (successful-clustering yes)
  (concept-card $?concept ?card)
=>
  (assert
    (print-concept $?concept))
  (printout t t t "instances of the concept " (list$ ?concept) " are:"))

(defrule print-clustering-results-2
  (declare (salience -70))
  (print-concept ?concept)
  (member-of ?concept ?event)
=>
  (printout t t ?event))

(defrule administration-1
  (declare (salience -72))
  ?x <- (print-concept ?concept)
=>
  (retract ?x))

(defrule max-level-of-hierarchy-1
  (declare (salience -79))
  (trigger-5 on)
  (successful-clustering yes)

```

```

=> (concept-length ?concept ?length)
    (assert
      (temp-concept-length ?concept ?length)))

(defrule max-level-of-hierarchy-2
  (declare (salience -80))
  (trigger-5 on)
  (successful-clustering yes)
  ?x <- (count-level ?level)
  ?y <- (temp-concept-length ?concept ?length)
  (test
    (> ?length ?level))
=> (retract ?x ?y)
    (assert
      (count-level ?length)))

(defrule print-characterization-message
  (declare (salience -83))
  (trigger-5 on)
  (successful-clustering yes)
=> (printout t t t t t "CHARACTERIZATION PROCESS" t "*****"
    "*****"))

... *****
... *
... *
... *           Characterization Process
... *
... *****
...
... *
... * Preparatory Computation
... *****
... *
... *

(defrule prepare-concept-attributes
  (declare (salience -85))
  (successful-clustering yes)
  (trigger-5 on)
  ?x <- (attribute ?schema (?att ?val ?rel))
=> (retract ?x)
    (assert
      (concept-attribute ?schema (?att ?val ?rel))))

(defrule administration-2
  (declare (salience -88))
  (successful-clustering yes)
  ?x <- (trigger-5 on)
=> (retract ?x)

```

```

(assert
  (trigger-7 on)))

(defrule administration-3a
  (declare (salience -89))
  ?x <- (trigger-7 on)
=>
  (retract ?x)
  (assert
    (trigger-0 on)))

(defrule event-att-val-copy
  (declare (salience -90))
  (trigger-0 on)
  (member-of ?concept ?event)
  (concept-attribute ?event (?att ?val ?rel))
=>
  (assert
    (att-val-cp ?att ?event ?val ?rel)))

(defrule initial-total-attribute-relevance
  (declare (salience -95))
  (trigger-0 on)
  (member-of ?concept ?event)
  (concept-attribute ?event (?att ?val ?rel))
  (not (concept-att-val ?concept ?att ?val ?total-rel))
=>
  (assert
    (concept-att-val ?concept ?att ?val 0.0)))

(defrule administration-3b
  (declare (salience -98))
  ?x <- (trigger-0 on)
=>
  (retract ?x)
  (assert
    (trigger-8 on)
    (trigger-9 on)
    (trigger-12 on)))

... *
... * Compute the Relevance of the (Att, Val) Pair
... * in the Concept Description
... *****
... *
... *

(defrule compute-total-attribute-relevance
  (declare (salience -100))
  (trigger-8 on)
  (trigger-9 on)
  ?x <- (concept-att-val ?concept ?att ?val ?total-rel)
  (member-of ?concept ?event)
  ?y <- (att-val-cp ?att ?event ?val ?rel)

```

```

=>
  (retract ?x ?y)
  (assert
    (concept-att-val ?concept ?att ?val =(total-rel + ?rel))))

(defrule concept-characterization
  (declare (salience -105))
  (trigger-8 on)
  (trigger-9 on)
  (go-print1 ?concept)
  (concept-card ?concept ?card)
  ?x <- (concept-att-val ?concept ?att ?val ?total-rel)
=>
  (retract ?x)
  (assert
    (concept-attribute ?concept (?att ?val =(total-rel / ?card))))

(defrule go-print-1
  (declare (salience -108))
  (trigger-8 on)
  (trigger-9 on)
  (concept-card $?concept ?card)
=>
  (assert
    (go-print1 $?concept))
  (printout t t t "description of the concept " (list$ ?concept) ":",))

(defrule go-print-2
  (declare (salience -106))
  (go-print1 ?concept)
  (concept-attribute ?concept (?att ?val ?rel))
=>
  (printout t t "(" ?att " " ?val " " ?rel ")",))

(defrule go-print-3
  (declare (salience -107))
  ?x <- (go-print1 ?concept)
=>
  (retract ?x))

;;; *
;;; * Refine the Concept Description:
;;; *   Remove the (Att, Val) Pairs With a Low Relevance
;;; *   *****
;;; *
(defrule threshold-remove-2
  (declare (salience -109))
  (trigger-8 on)
  (trigger-9 on)
  ?x <- (concept-attribute $?concept (?att ?val ?rel))
  (threshold concept-att ?th-val)
  (test

```

```

=> (< ?rel ?th-val))

(retract ?x)
(printout t t t t "removed (" ?att " " ?val " " ?rel ") triplet"
  t " from the concept (" (list$ ?concept) ") description:"
  t " low attribute relevance"))

;;; *
;;; * Determine Both Instances and
;;; * Hierarchical Links of a Concept
;;; *****
;;; *
;;; *

(defrule concept-instances-determination
  (declare (salience -110))
  (trigger-8 on)
  (trigger-9 on)
  (successful-clustering yes)
  (concept-card ?concept ?card)
  (member-of ?concept ?event)
=>
  (assert
    (instance-link-sum-strength ?concept ?event 0.0)
    (instance ?concept (?event 0.0))
    (instance-of-concept ?event (?concept 0.0))))

(defrule concept-specialization-generalization
  (declare (salience -110))
  (trigger-8 on)
  (trigger-9 on)
  (not (successful-clustering ?yes-no))
  (concept-card ?concept ?card)
  (member-of ?concept ?event)
=>
  (assert
    (instance-link-sum-strength ?concept ?event 0.0)
    (subordinate-concept ?concept (?event 0.0))
    (superordinate-concept ?event (?concept 0.0))))

;;; *
;;; * Compute the Strength of the 'Instance' and
;;; * 'Hierarchical' Links
;;; *****
;;; *
;;; *

(defrule find-participating-attributes-1
  (declare (salience -112))
  (trigger-8 on)
  (trigger-9 on)
  (subordinate-concept ?concept (?event 0.0))
  (concept-attribute ?concept (?att ?val ?rel1))
  (concept-attribute ?event (?att ?val ?rel2))

```

```

=>
  (assert
    (partic-att-val ?concept ?event ?att ?val =(?rel1 * ?rel2))))

(defrule find-participating-attributes-2
  (declare (salience -112))
  (trigger-8 on)
  (trigger-9 on)
  (instance ?concept (?event 0.0))
  (concept-attribute ?concept (?att ?val ?rel1))
  (concept-attribute ?event (?att ?val ?rel2))
=>
  (assert
    (partic-att-val ?concept ?event ?att ?val =(?rel1 * ?rel2))))

(defrule administration-4
  (declare (salience -114))
  ?x <- (trigger-9 on)
=>
  (retract ?x)
  (assert
    (trigger-10 on)))

(defrule total-instance-link-strength
  (declare (salience -115))
  (trigger-8 on)
  (trigger-10 on)
  ?x <- (partic-att-val ?concept ?event ?att ?val ?mult-rel)
  ?y <- (instance-link-sum-strength ?concept ?event ?sum)
=>
  (retract ?x ?y)
  (assert
    (instance-link-sum-strength ?concept ?event =(?sum + ?mult-rel))))

(defrule concept-att-val-copy
  (declare (salience -115))
  (trigger-8 on)
  (trigger-10 on)
  (member-of ?concept ?event)
  (concept-attribute ?concept (?att ?val ?rel))
=>
  (assert
    (concept-att-val-copy ?att ?concept ?event ?val)))

(defrule initial-concept-att-card
  (declare (salience -115))
  (trigger-8 on)
  (trigger-10 on)
  (concept-card ?concept ?card)
  (member-of ?concept ?event)
=>
  (assert
    (concept-att-card ?concept ?event 0.0)))

```

```

(defrule compute-concept-att-card
  (declare (salience -118))
  (trigger-8 on)
  (trigger-10 on)
  (member-of ?concept ?event)
  ?x <- (concept-att-val-copy ?att ?concept ?event ?val)
  ?y <- (concept-att-card ?concept ?event ?att-card)
=>
  (retract ?x ?y)
  (assert
    (concept-att-card ?concept ?event =( ?att-card + 1))))

(defrule compute-instance-link-strength
  (declare (salience -120))
  (trigger-8 on)
  (trigger-10 on)
  (go-print2 ?concept)
  ?x <- (instance ?concept (?event 0.0))
  ?y <- (instance-of-concept ?event (?concept 0.0))
  ?z <- (instance-link-sum-strength ?concept ?event ?total-strength)
  ?w <- (concept-att-card ?concept ?event ?att-card)
=>
  (retract ?x ?y ?z ?w)
  (assert
    (instance ?concept (?event =( ?total-strength / ?att-card)))
    (instance-of-concept ?event (?concept =( ?total-strength / ?att-card)))))

(defrule compute-specialization-link-strength
  (declare (salience -120))
  (trigger-8 on)
  (trigger-10 on)
  (go-print2 ?concept)
  ?x <- (subordinate-concept ?concept (?event 0.0))
  ?y <- (superordinate-concept ?event (?concept 0.0))
  ?z <- (instance-link-sum-strength ?concept ?event ?total-strength)
  ?w <- (concept-att-card ?concept ?event ?att-card)
=>
  (retract ?x ?y ?z ?w)
  (assert
    (subordinate-concept ?concept (?event =( ?total-strength / ?att-card)))
    (superordinate-concept ?event (?concept =( ?total-strength / ?att-card)))))

(defrule go-print-4
  (declare (salience -124))
  (trigger-8 on)
  (trigger-10 on)
  (concept-card $?concept ?card)
=>
  (assert
    (go-print2 $?concept))
  (printout t t t t "hierarchical links of the concept " (list$ ?concept) ":"))

(defrule go-print-5
  (declare (salience -121))

```

```

      (go-print2 ?concept)
      (instance ?concept (?event ?strength))
=>
      (printout t t "(INSTANCE (\" ?event \" \" ?strength \"))"))

(defrule go-print-6
  (declare (salience -121))
  (go-print2 ?concept)
  (subordinate-concept ?concept (?sub-concept ?strength))
=>
  (printout t t "(SUBORDINATE-CONCEPT (\" ?sub-concept \" \" ?strength \"))"))

(defrule go-print-7
  (declare (salience -122))
  ?x <- (go-print2 ?concept)
=>
  (retract ?x))

(defrule administration-5
  (declare (salience -125))
  (trigger-8 on)
  ?x <- (trigger-10 on)
=>
  (retract ?x))

;;; *
;;; * Refine the Concept Description:
;;; *   Climb the Hierarchy
;;; * *****
;;; *

;;;(defrule climbing-domain-hierarchy-2
;;;  (declare (salience -130))
;;;  (trigger-8 on)
;;;  (concept-card ($?concept) ?card)
;;;  (concept-attribute ($?concept) (?att ?val ?rel))
;;;  (infer ($?concept) (?att ?val ?rel1))
;;;=>
;;;  (printout t t t "remove from the concept (\" (list$ ?concept)
;;;    \") description\" t \" the (att, val) pairs covered by (\" ?att \" \" ?val \"))"))

;;; *
;;; * Refine the Concept Description:
;;; *   Remove the Links With a Low Strength
;;; * *****
;;; *

(defrule instance-threshold-remove
  (declare (salience -140))
  (trigger-8 on)
  (concept-card $?concept ?card)
  ?x <- (instance $?concept (?inst ?rel))
  (threshold instance-link ?th-val)

```

```

(test
  (<= ?rel ?th-val))
=>
(retract ?x)
(printout t t t t "removed instance " ?inst t " from the concept "
  (list$ ?concept) " description: low link strength"))

(defrule instance-of-concept-threshold-remove
  (declare (salience -140))
  (trigger-8 on)
  (concept-card $?concept ?card)
  (member-of $?concept ?event)
  ?x <- (instance-of-concept ?event ($?concept ?rel))
  (threshold instance-of-link ?th-val)
  (test
    (<= ?rel ?th-val))
=>
(retract ?x)
(printout t t t t "removed link to the concept " (list$ ?concept)
  t " from the event " ?event " description: low link strength"))

(defrule subordinate-concept-threshold-remove
  (declare (salience -140))
  (trigger-8 on)
  (concept-card $?concept ?card)
  ?x <- (subordinate-concept $?concept ($?inst ?rel))
  (threshold subordinate-link ?th-val)
  (test
    (<= ?rel ?th-val))
=>
(retract ?x)
(printout t t t t "removed subordinate concept " (list$ ?inst)
  t " from the concept " (list$ ?concept) " description: low link strength"))

(defrule superordinate-concept-threshold-remove
  (declare (salience -140))
  (trigger-8 on)
  (concept-card $?concept ?card)
  (member-of $?concept ($?event))
  ?x <- (superordinate-concept ($?event) ($?concept ?rel))
  (threshold superordinate-link ?th-val)
  (test
    (<= ?rel ?th-val))
=>
(retract ?x)
(printout t t t t "removed link to the concept " (list$ ?concept)
  t " from the event (" (list$ ?event) ") description: low link strength"))

... *
... *
... * Print the Result of the Characterization Process
... * (Concept Descriptions)
... *****
... *
... *

```

```

(defrule define-concepts-to-be-printed
  (declare (salience -160))
  (trigger-8 on)
  (trigger-12 on)
  (concept-card $?concept ?card)
=>
  (assert
    (print-concept-description $?concept))
  (printout t t t "description of the concept " (list$ ?concept) "："))

(defrule print-a-concept-1
  (declare (salience -150))
  (trigger-8 on)
  (print-concept-description ?concept)
  (concept-attribute ?concept (?att ?val ?rel))
=>
  (printout t t "(concept-attribute (" ?att " " ?val " " ?rel "))"))

(defrule print-a-concept-2
  (declare (salience -150))
  (trigger-8 on)
  (print-concept-description ?concept)
  (instance ?concept (?inst ?rel))
=>
  (printout t t "(instance (" ?inst " " ?rel "))"))

(defrule print-a-concept-3
  (declare (salience -150))
  (trigger-8 on)
  (print-concept-description ?concept)
  (instance-of-concept ?concept ($?sup-concept ?rel))
=>
  (printout t t "(instance-of-concept (" (list$ ?sup-concept) " " ?rel "))"))

(defrule print-a-concept-4
  (declare (salience -150))
  (trigger-8 on)
  (print-concept-description ?concept)
  (subordinate-concept ?concept ($?sub-concept ?rel))
=>
  (printout t t "(subordinate-concept (" (list$ ?sub-concept) " " ?rel "))"))

(defrule print-a-concept-5
  (declare (salience -150))
  (trigger-8 on)
  (print-concept-description ?concept)
  (superordinate-concept ?concept ($?sup-concept ?rel))
=>
  (printout t t "(superordinate-concept (" (list$ ?sup-concept) " " ?rel "))"))

;;; *
;;; * Garbage Collection
;;; *****

```

```

;;; *

(defrule garbage-collection-3
  (declare (salience -170))
  (trigger-8 on)
  ?x <- (print-concept-description ?concept)
=>
  (retract ?x))

(defrule garbage-collection-4
  (declare (salience -190))
  (trigger-8 on)
  ?x <- (top-relevance goal-rel-att ?att ?rel)
=>
  (retract ?x))

(defrule garbage-collection-5
  (declare (salience -190))
  (trigger-8 on)
  (trigger-12 on)
  ?x <- (concept-card ?concept ?card)
=>
  (retract ?x))

(defrule garbage-collection-6
  (declare (salience -190))
  (trigger-8 on)
  (trigger-12 on)
  ?x <- (number-of-concepts ?number&~0)
=>
  (retract ?x)
  (assert
    (number-of-concepts 0)))

(defrule garbage-collection-7
  (declare (salience -190))
  (trigger-8 on)
  (trigger-12 on)
  ?x <- (successful-clustering ?yes-no)
=>
  (retract ?x))

(defrule remove-old-attributes
  (declare (salience -191))
  (trigger-8 on)
  (trigger-12 on)
  (count-level ?level)
  (concept-length ?concept ?level)
  (member-of ?concept ?event)
  ?x <- (attribute ?event (?att ?val ?rel))
=>
  (retract ?x))

(defrule garbage-collection-8

```

```

      (declare (salience -192))
      (trigger-8 on)
      ?x <- (trigger-12 on)
=>
      (retract ?x))

... *****
... *
... *
... *      Construct the Hierarchy of Concepts
... *
... *
... *****

(defrule climbing-concept-hierarchy-1
  (declare (salience -193))
  (trigger-8 on)
  (not (count-level 0))
  ?x <- (do-it-again no)
=>
  (retract ?x)
  (assert
    (trigger-11 on)))

(defrule print-hierarchy-message
  (declare (salience -194))
  (trigger-8 on)
  (trigger-11 on)
=>
  (printout t t t t t "BUILDING A HIERARCHY" t "*****"))

(defrule climbing-concept-hierarchy-2
  (declare (salience -196))
  (trigger-8 on)
  (trigger-11 on)
  (count-level ?level)
  (concept-length ($?concept1 ?concept2) ?level)
=>
  (assert
    (member-of ($?concept1) ($?concept1 ?concept2))))

(defrule climbing-concept-hierarchy-3
  (declare (salience -197))
  (trigger-8 on)
  (trigger-11 on)
  (count-level ?level)
  (concept-length ?concept ?level)
  ?x <- (member-of ?concept ?event)
=>
  (retract ?x))

(defrule climbing-concept-hierarchy-4
  (declare (salience -199))
  ?x <- (trigger-11 on)
=>

```

```

    (retract ?x)
    (assert
      (do-it-again yes)
      (trigger-14 on)))

(defrule member-info-copy-2
  (declare (salience -200))
  (do-it-again yes)
  (trigger-14 on)
  (count-level ?level)
  (concept-length ($?concept1) =(?level - 1))
  (member-of ($?concept1) ($?concept1 ?concept2))
=>
  (assert
    (temp-member-info-2 ($?concept1) ($?concept1 ?concept2))))

(defrule concept-info-2
  (declare (salience -201))
  (do-it-again yes)
  (trigger-14 on)
  (count-level ?level)
  (concept-length ($?concept1) =(?level - 1))
  (member-of ($?concept1) ($?concept1 ?concept2))
  (not (concept-card ($?concept1) ?val))
=>
  (assert
    (concept-card ($?concept1) 0.0)
    (temp-concept-6 ($?concept1))))

(defrule concept-cardinality-2
  (declare (salience -202))
  (do-it-again yes)
  (trigger-14 on)
  ?x <- (concept-card ?concept ?val)
  ?y <- (temp-member-info-2 ?concept ?event)
=>
  (retract ?x ?y)
  (assert
    (concept-card ?concept =(?val + 1))))

(defrule climbing-concept-hierarchy-5
  (declare (salience -203))
  ?x <- (trigger-14 on)
  ?y <- (count-level ?level)
=>
  (retract ?x ?y)
  (assert
    (count-level =(?level - 1))))

(defrule print-level-2
  (declare (salience -204))
  (do-it-again yes)
  (count-level ?level)
=>

```

```

(printout t t t t "LEVEL " ?level))

(defrule concept-number-6
  (declare (salience -205))
  ?x <- (number-of-concepts ?val)
  ?y <- (temp-concept-6 ?concept)
=>
  (retract ?x ?y)
  (assert
    (number-of-concepts =(?val + 1))))

(defrule concept-number-7
  (declare (salience -206))
  (do-it-again yes)
  (number-of-concepts ?number)
=>
  (printout t " - number of concepts evaluated at this level: " ?number ))

(defrule climbing-concept-hierarchy-6
  (declare (salience -208))
  (trigger-8 on)
  ?x <- (do-it-again yes)
=>
  (retract ?x)
  (assert
    (trigger-7 on)
    (do-it-again no)))

... *****
;;; *
;;; *
;;; *           End of the Process
;;; *
;;; *
;;; *****

(defrule end
  (declare (salience *minimum-salience*))
  (trigger-8 on)
  ?x <- (do-it-again no)
=>
  (retract ?x)
  (printout t t t t "process is FINISHED." t t))

```

APPENDIX B

THE UPDATE RULES

```
(defrule administration-1
  (declare (salience -72))
  ?x <- (print-concept ?concept)
=>
  (retract ?x)
  (assert
    (trigger-16 on)))

(defrule mark-atts-for-update-2
  (declare (salience -80))
  (successful-clustering yes)
  (trigger-16 on)
  (list goal-rel-att (?att ?val ?rel))
  (number-of-concepts-at-level ?att 0.0)
  (not (number-of-concepts-at-level ?att ?number&~0.0))
=>
  (assert
    (update ?att ?val negative))
  (printout t t t t "=>decrease the strength of the rule that posted the triplet"
    t " (" ?att " " ?val " " ?rel ")"))

(defrule mark-atts-for-update-3
  (declare (salience -80))
  (successful-clustering yes)
  (trigger-16 on)
  (list goal-rel-att (?att ?val ?rel))
  (exists
    (number-of-concepts-at-level ?att ?number&~0.0))
=>
  (assert
    (update ?att ?val positive))
  (printout t t t t "=>increase the strength of the rule that posted the triplet"
    t " (" ?att " " ?val " " ?rel ")"))

(defrule trigger-16-off
  (declare (salience -81))
  (successful-clustering yes)
  ?x <- (trigger-16 on)
=>
  (retract ?x))
```

```

(defrule update-1
  (declare (salience -82))
  (successful-clustering yes)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  ?y <- (update ?att ?val positive)
  (list-length goal-rel-att ?length)
=>
  (retract ?x ?y)
  (assert
    (list goal-rel-att (?att ?val =(?rel + (?rel / ?length))))))

(defrule update-2
  (declare (salience -82))
  (successful-clustering yes)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  ?y <- (update ?att ?val negative)
  (list-length goal-rel-att ?length)
=>
  (retract ?x ?y)
  (assert
    (list goal-rel-att (?att ?val =(?rel - (?rel / ?length))))))

(defrule update-3
  (declare (salience -83))
  (successful-clustering yes)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  (test
    (> ?rel 1.0))
=>
  (retract ?x)
  (assert
    (list goal-rel-att (?att ?val 1.0))))

(defrule update-4
  (declare (salience -83))
  (successful-clustering yes)
  ?x <- (list goal-rel-att (?att ?val ?rel))
  (test
    (< ?rel 0.0))
=>
  (retract ?x)
  (assert
    (list goal-rel-att (?att ?val 0.0))))

(defrule print-update
  (declare (salience -84))
  (successful-clustering yes)
=>
  (printout t t t t "new goal-rel-att list:"))

(defrule print-update-list
  (declare (salience -85))
  (successful-clustering yes)
  (list goal-rel-att (?att ?val ?rel))

```

=> (printout t t " (" ?att " " ?val " " ?rel ")"))

APPENDIX C

THE STRUCTURE OF THE ATTRIBUTE DOMAINS:

AN EXAMPLE

```
;;; *
;;; * Attribute: Function
;;; *****
;;; *
;;; *

;;; *** Printing ***

(defrule infer-attribute-value-relevance-triplet-26
  (infer ?event (function printing ?rel))
  <=
    (infer ?event (function file-printing ?rel))
    (= >
      (assert
        (flag-triplet ?event (function file-printing ?rel))))))

(defrule infer-attribute-value-relevance-triplet-27
  (infer ?event (function printing ?rel))
  <=
    (infer ?event (function off-line-printing ?rel))
    (= >
      (assert
        (flag-triplet ?event (function off-line-printing ?rel))))))

;;; *** Editing ***

(defrule infer-attribute-value-relevance-triplet-28
  (infer ?event (function editing ?rel))
  <=
    (infer ?event (function text-editing ?rel))
    (= >
      (assert
        (flag-triplet ?event (function text-editing ?rel))))))

(defrule infer-attribute-value-relevance-triplet-29
  (infer ?event (function editing ?rel))
  <=
    (infer ?event (function screen-oriented-editor ?rel))
    (= >
      (assert
        (flag-triplet ?event (function screen-oriented-editor ?rel))))))
```

```

;;; *** File Manipulation ***

(defrule infer-attribute-value-relevance-triplet-30
  (infer ?event (function file-manipulation ?rel))
  <=
    {infer ?event (function catenate-and-print ?rel)}
    {=>
      (assert
        (flag-triplet ?event (function catenate-and-print ?rel))))))

(defrule infer-attribute-value-relevance-triplet-38
  (infer ?event (function file-manipulation ?rel))
  <=
    {infer ?event (function text-formating ?rel)}
    {=>
      (assert
        (flag-triplet ?event (function text-formating ?rel))))))

(defrule infer-attribute-value-relevance-triplet-39
  (infer ?event (function file-manipulation ?rel))
  <=
    {infer ?event (function search-for-a-pattern ?rel)}
    {=>
      (assert
        (flag-triplet ?event (function search-for-a-pattern ?rel))))))

;;; *** Text Formating ***

(defrule infer-attribute-value-relevance-triplet-52
  (infer ?event (function text-formating ?rel))
  <=
    {infer ?event (function text-formating-and-typesetting ?rel)}
    {=>
      (assert
        (flag-triplet ?event (function text-formating-and-typesetting ?rel))))))

(defrule infer-attribute-value-relevance-triplet-105
  (infer ?event (function text-formating ?rel))
  <=
    {infer ?event (function phototypesetter-simulator ?rel)}
    {=>
      (assert
        (flag-triplet ?event (function phototypesetter-simulator ?rel))))))

;;; *
;;; * Attribute: Domain
;;; *****
;;; *
;;; *

;;; *** File ***

(defrule infer-attribute-value-relevance-triplet-162

```

```

      (infer ?event (domain file ?rel))
<=
  (infer ?event (domain user-type-file ?rel))
  {=>
    (assert
      (flag-triplet ?event (domain user-type-file ?rel))))))

(defrule infer-attribute-value-relevance-triplet-163
  (infer ?event (domain file ?rel))
<=
  (infer ?event (domain system-type-file ?rel))
  {=>
    (assert
      (flag-triplet ?event (domain system-type-file ?rel))))))

;;; *
;;; * Attribute: Range
;;; *****
;;; *
;;;
;;;
;;; *** File ***

(defrule infer-attribute-value-relevance-triplet-211
  (infer ?event (range file ?rel))
<=
  (infer ?event (range source-file ?rel))
  {=>
    (assert
      (flag-triplet ?event (range source-file ?rel))))))

(defrule infer-attribute-value-relevance-triplet-212
  (infer ?event (range file ?rel))
<=
  (infer ?event (range user-file ?rel))
  {=>
    (assert
      (flag-triplet ?event (range user-file ?rel))))))

(defrule infer-attribute-value-relevance-triplet-222
  (infer ?event (range file ?rel))
<=
  (infer ?event (range formatted-file ?rel))
  {=>
    (assert
      (flag-triplet ?event (range formatted-file ?rel))))))

```

APPENDIX D

ATTRIBUTES USED IN COMMAND DESCRIPTIONS

General-Type Attributes

1. function (primary function)
2. is-a (purpose)
 - general-utility: 1
 - communication-with-other-systems: 1C
 - graphics-and-computer-aided-design: 1G
3. related-instructions (see-also, associative links)
4. domain
 - manual
 - user-file
 - etc.
5. range
 - input-dependent
 - user-file
 - etc.
6. type-of-parameters
 - file-name
 - keyword
 - etc.

7. number-of-nonoptional-parameters
 - 0,1,2,...,100 (as many as needed)
8. number-of-optional-parameters
 - 0,1,2,...
9. processing-time
 - input-dependent
 - very-short (immediate response)
 - short (few seconds)
 - medium (up to a minute)
 - long (few minutes)
 - very-long (one or more hours)
10. input-device
 - output-of-another-operation
 - file
 - standard-input (default: keyboard)
 - etc.
11. output-device
 - input-dependent
 - printer
 - magnetic-tape
 - standard-output (default: terminal-screen)
 - etc.
12. meaningful-mnemonic
 - yes

- no
- 13. number-of-flags
- 14. instance-of (instance of the same command with no flags used)

Special-Purpose Attributes

- 15. intermediate-storage
 - virtual-memory
 - temporary-file
 - etc.
- 16. warning-diagnostics
 - suppressed
- 17. arranged-output
 - numbered-lines
 - n-columns
 - n-lines
 - limited-lines
 - nonprinting-characters-displayed
 - summary-of-statistics
 - formatted-file
 - indented-output
- 18. output-file
 - specified
 - output file name: output

- standard-output

19. source-file

- directory
- terminal

20. output-format

- long
- short

APPENDIX E

SELECTED COMMANDS

- | | |
|------------------------------|-------------------------------|
| 1. <code>appropos</code> | 24. <code>du</code> |
| 2. <code>as (v,w,t)</code> | 25. <code>ed</code> |
| 3. <code>at</code> | 26. <code>efl</code> |
| 4. <code>biff</code> | 27. <code>ex</code> |
| 5. <code>cal</code> | 28. <code>f77 (o,w)</code> |
| 6. <code>calendar</code> | 29. <code>file</code> |
| 7. <code>cat (n,v)</code> | 30. <code>find</code> |
| 8. <code>cc (c,w,o)</code> | 31. <code>finger (l,s)</code> |
| 9. <code>cd</code> | 32. <code>fmt</code> |
| 10. <code>checknr (c)</code> | 33. <code>fpr</code> |
| 11. <code>chfn</code> | 34. <code>from</code> |
| 12. <code>chmod</code> | 35. <code>fsplit</code> |
| 13. <code>clear</code> | 36. <code>graph</code> |
| 14. <code>cmp</code> | 37. <code>grep (n)</code> |
| 15. <code>colert</code> | 38. <code>iostat</code> |
| 16. <code>cp (r)</code> | 39. <code>kill</code> |
| 17. <code>date</code> | 40. <code>last (N)</code> |
| 18. <code>dbx (i)</code> | 41. <code>lastcomm</code> |
| 19. <code>dc</code> | 42. <code>learn</code> |
| 20. <code>dd</code> | 43. <code>leave</code> |
| 21. <code>deroff</code> | 44. <code>lex (t,v)</code> |
| 22. <code>diction</code> | 45. <code>lint (u)</code> |
| 23. <code>diff (l)</code> | 46. <code>lisp</code> |

47. liszt (o,w,T)	74. ps
48. ln	75. pwd
49. lock	76. px
50. login	77. quota
51. lpq	78. rm
52. lpr (p,i)	79. rmdir
53. lprm	80. script
54. ls (l)	81. size
55. mail	82. sleep
56. mail-user	83. sort (o)
57. man	84. spell
58. mkdir	85. split
59. more (n)	86. struct
60. msgs	87. stty
61. mt	88. style
62. mv	89. sum
63. nice	90. sysline
64. nroff	91. tabs
65. passwd	92. tail
66. pc (w)	93. talk
67. pi (w)	94. tc
68. pix (w)	95. time
69. plot	96. touch
70. pmerge	97. troff (t)
71. pr (n,ln)	98. tty
72. print	99. uptime
73. prmail	100. users

- 101. vi
- 102. vlp
- 103. vmstat
- 104. w
- 105. wait
- 106. wall
- 107. what
- 108. whatis
- 109. whereis
- 110. which
- 111. who
- 112. whoami
- 113. write
- 114. xsend
- 115. xget

APPENDIX F

SAMPLE RUN 1:

CURRENT CONTEXT

=> run

CLUSTERING PROCESS

there are 14.0 events in the initial set

inferred (FUNCTION EDITING 1.0) triplet
in the description of VI

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of TROFF-T

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of TROFF

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of STYLE

inferred (FUNCTION PRINTING 1.0) triplet
in the description of SPIT-I

inferred (FUNCTION PRINTING 1.0) triplet
in the description of PRINT

inferred (FUNCTION PRINTING 1.0) triplet
in the description of PR

inferred (FUNCTION PRINTING 1.0) triplet
in the description of LPR

inferred (FUNCTION EDITING 1.0) triplet
in the description of EX

inferred (FUNCTION EDITING 1.0) triplet
in the description of ED

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION EDITING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION PRINTING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION FILE-PRINTING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION TEXT-FORMATING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION FIND-SPELLING-ERRORS):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION SIGN-ON):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION SET-TERMINAL-OPTIONS):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (DOMAIN USER-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (RANGE USER-FILE):

climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (RANGE PRINTED-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (INPUT-DEVICE STANDARD-INPUT):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (OUTPUT-DEVICE LINE-PRINTER):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (OUTPUT-DEVICE LASER-PRINTER):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0):
climbing a domain hierarchy

removed (FUNCTION FILE-PRINTING 0.9) from the goal-rel-att list:
already covered

removed (FUNCTION SET-TERMINAL-OPTIONS 0.05) from the goal-rel-att list:
low relevance

removed (FUNCTION SIGN-ON 0.2) triplet from the goal-rel-att list:
doesn't cover any event

number of generated goal-related (att, val) pairs = 11

LEVEL 0 - ATTEMPT 1

concept: (C-1) - classifying attribute: FUNCTION

following concept is defined by the attribute FUNCTION
and value EDITING:
(C-2) with an instance ED
(C-2) with an instance EX
(C-2) with an instance VI

following concept is defined by the attribute FUNCTION
and value PRINTING:

- (C-3) with an instance LPR
- (C-3) with an instance PR
- (C-3) with an instance PRINT
- (C-3) with an instance SPIT-I

following concept is defined by the attribute FUNCTION
and value TEXT-FORMATING:

- (C-4) with an instance NROFF
- (C-4) with an instance STYLE
- (C-4) with an instance TROFF
- (C-4) with an instance TROFF-T

following concept is defined by the attribute FUNCTION
and value FIND-SPELLING-ERRORS:

- (C-5) with an instance SPELL

following concept is defined by the attribute FUNCTION
and value OTHER:

- (C-6) with an instance CAT
- (C-6) with an instance STTY

there are 5.0 concepts

- cardinality of the concept (C-2) is 3.0
- cardinality of the concept (C-3) is 4.0
- cardinality of the concept (C-4) is 4.0
- cardinality of the concept (C-5) is 1.0
- cardinality of the concept (C-6) is 2.0

heuristic criterion DISTRIBUTION-OF-EVENTS is going to be applied

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 1 - ATTEMPT 2

concept: (C-3) - classifying attribute: DOMAIN

following concept is defined by the attribute DOMAIN
and value USER-FILE:

following concept is defined by the attribute DOMAIN
and value OTHER:

(C-7) with an instance LPR
(C-7) with an instance PR
(C-7) with an instance PRINT
(C-7) with an instance SPIT-I

there are 5.0 concepts

cardinality of the concept (C-7) is 4.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 2 - ATTEMPT 3

concept: (C-7) - classifying attribute: RANGE

following concept is defined by the attribute RANGE
and value USER-FILE:

following concept is defined by the attribute RANGE
and value PRINTED-FILE:

(C-8) with an instance LPR
(C-8) with an instance PR
(C-8) with an instance PRINT
(C-8) with an instance SPIT-I

following concept is defined by the attribute RANGE
and value OTHER:

there are 5.0 concepts

cardinality of the concept (C-8) is 4.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering ==> initialization process

LEVEL 3 - ATTEMPT 4

concept: (C-8) - classifying attribute: INPUT-DEVICE

following concept is defined by the attribute INPUT-DEVICE
and value STANDARD-INPUT:

(C-9) with an instance LPR
(C-9) with an instance PR
(C-9) with an instance PRINT
(C-9) with an instance SPIT-I

following concept is defined by the attribute INPUT-DEVICE
and value OTHER:

there are 5.0 concepts
cardinality of the concept (C-9) is 4.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering ==> initialization process

LEVEL 4 - ATTEMPT 5

concept: (C-9) - classifying attribute: OUTPUT-DEVICE

following concept is defined by the attribute OUTPUT-DEVICE
and value LINE-PRINTER:

(C-10) with an instance LPR
(C-10) with an instance PRINT

following concept is defined by the attribute OUTPUT-DEVICE
and value LASER-PRINTER:

(C-11) with an instance SPIT-I

following concept is defined by the attribute OUTPUT-DEVICE
and value OTHER:

(C-12) with an instance PR

there are 7.0 concepts

cardinality of the concept (C-12) is 1.0

cardinality of the concept (C-11) is 1.0

cardinality of the concept (C-10) is 2.0

cardinality of the concept (C-2) is 3.0

cardinality of the concept (C-4) is 4.0

cardinality of the concept (C-5) is 1.0

cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.0

heuristic criterion DISTRIBUTION-OF-EVENTS is satisfied

clustering process is COMPLETED SUCCESSFULLY

instances of the concept (C-6) are:

CAT
STTY

instances of the concept (C-5) are:

SPELL

instances of the concept (C-4) are:

NROFF
STYLE
TROFF
TROFF-T

instances of the concept (C-2) are:

ED
EX
VI

instances of the concept (C-10) are:

LPR
PRINT

instances of the concept (C-11) are:

SPIT-I

instances of the concept (C-12) are:

PR

CHARACTERIZATION PROCESS

description of the concept (C-6):

(NUMBER-OF-FLAGS 11 0.5)
 (OUTPUT-DEVICE PRIMARY-MEMORY 0.5)
 (PROCESSING-TIME VERY-SHORT 0.5)
 (NUMBER-OF-OPTIONAL-PARAMETERS 0 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.5)
 (RANGE TERMINAL-CONTROL-INFO 0.5)
 (DOMAIN TERMINAL-CONTROL-INFO 0.5)
 (FUNCTION SET-TERMINAL-OPTIONS 0.5)
 (NUMBER-OF-FLAGS 4 0.5)
 (MEANINGFUL-MNEMONIC YES 1.0)
 (OUTPUT-DEVICE STANDARD-OUTPUT 0.5)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME MEDIUM 0.5)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.5)
 (TYPE-OF-PARAMETERS USER-FILE 0.5)
 (RANGE FILE 0.5)
 (DOMAIN USER-FILE 0.5)
 (FUNCTION CATENATE-AND-PRINT 0.5)

description of the concept (C-5):

(NUMBER-OF-FLAGS 6 1.0)
 (MEANINGFUL-MNEMONIC YES 1.0)
 (OUTPUT-DEVICE STANDARD-OUTPUT 1.0)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 1.0)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE LIST-OF-SPELLING-ERRORS 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FIND-SPELLING-ERRORS 1.0)

description of the concept (C-4):

(NUMBER-OF-FLAGS 14 0.5)
 (OUTPUT-DEVICE GRAPHICS-SYSTEMS-PHOTOTYPESETTER 0.5)
 (FUNCTION TEXT-FORMATING-AND-TYPESETTING 0.5)
 (NUMBER-OF-FLAGS 8 0.25)
 (MEANINGFUL-MNEMONIC YES 0.25)
 (OUTPUT-DEVICE STANDARD-OUTPUT 0.25)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.25)
 (RANGE ANALYSIS-RESULT-REPORT 0.25)
 (FUNCTION DOCUMENT-ANALYSIS 0.25)

(NUMBER-OF-FLAGS 10 0.25)
 (MEANINGFUL-MNEMONIC NO 0.75)
 (OUTPUT-DEVICE TYPEWRITER-LIKE-DEVICES 0.25)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.75)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE FORMATED-FILE 0.75)
 (DOMAIN FILE 1.0)
 (FUNCTION TEXT-FORMATING 1.0)

description of the concept (C-2):

(NUMBER-OF-FLAGS 5 0.3333333333)
 (PROCESSING-TIME MEDIUM 0.3333333333)
 (TYPE-OF-PARAMETERS FILE 0.3333333333)
 (RANGE FILE 0.3333333333)
 (DOMAIN FILE 0.3333333333)
 (FUNCTION SCREEN-ORIENTED-EDITOR 0.3333333333)
 (NUMBER-OF-FLAGS 6 0.3333333333)
 (MEANINGFUL-MNEMONIC NO 0.6666666665)
 (NUMBER-OF-OPTIONAL-PARAMETERS 0 0.6666666665)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.6666666665)
 (FUNCTION EDITING 1.0)
 (NUMBER-OF-FLAGS 2 0.3333333333)
 (MEANINGFUL-MNEMONIC YES 0.3333333333)
 (OUTPUT-DEVICE DISK 1.0)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME SHORT 0.6666666665)
 (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.3333333333)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.3333333333)
 (TYPE-OF-PARAMETERS USER-FILE 0.6666666665)
 (RANGE USER-FILE 0.6666666665)
 (DOMAIN USER-FILE 0.6666666665)
 (FUNCTION TEXT-EDITING 0.6666666665)

description of the concept (C-10):

(NUMBER-OF-FLAGS 0 0.5)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.5)
 (FUNCTION FILE-PRINTING 0.5)
 (FUNCTION PRINTING 1.0)
 (NUMBER-OF-FLAGS 20 0.5)
 (MEANINGFUL-MNEMONIC YES 1.0)
 (OUTPUT-DEVICE LINE-PRINTER 1.0)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.5)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)

(FUNCTION OFF-LINE-PRINTING 0.5)

description of the concept (C-11):

(FUNCTION PRINTING 1.0)
 (NUMBER-OF-FLAGS 6 1.0)
 (MEANINGFUL-MNEMONIC NO 1.0)
 (OUTPUT-DEVICE LASER-PRINTER 1.0)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 1 1.0)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 1.0)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FILE-PRINTING 1.0)

description of the concept (C-12):

(FUNCTION PRINTING 1.0)
 (NUMBER-OF-FLAGS 9 1.0)
 (MEANINGFUL-MNEMONIC YES 1.0)
 (OUTPUT-DEVICE PRINTER 1.0)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 1.0)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FILE-PRINTING 1.0)

removed (attribute, value) pairs with the relevance below the threshold

hierarchical links of the concept (C-6):

(INSTANCE (STTY 0.3157894737))
 (INSTANCE (CAT 0.3421052631))

hierarchical links of the concept (C-5):

(INSTANCE (SPELL 1.0))

hierarchical links of the concept (C-4):

(INSTANCE (TROFF-T 0.8125))
 (INSTANCE (TROFF 0.8125))
 (INSTANCE (STYLE 0.5))
 (INSTANCE (NROFF 0.6875))

hierarchical links of the concept (C-2):

(INSTANCE (VI 0.4545454546))

(INSTANCE (EX 0.757575758))
 (INSTANCE (ED 0.575757576))

hierarchical links of the concept (C-10):

(INSTANCE (PRINT 0.625))
 (INSTANCE (LPR 0.625))

hierarchical links of the concept (C-11):

(INSTANCE (SPIT-I 1.0))

hierarchical links of the concept (C-12):

(INSTANCE (PR 1.0))

description of the concept (C-6):

(concept-attribute (FUNCTION CATENATE-AND-PRINT 0.5))
 (concept-attribute (DOMAIN USER-FILE 0.5))
 (concept-attribute (RANGE FILE 0.5))
 (concept-attribute (TYPE-OF-PARAMETERS USER-FILE 0.5))
 (concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.5))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5))
 (concept-attribute (PROCESSING-TIME MEDIUM 0.5))
 (concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
 (concept-attribute (OUTPUT-DEVICE STANDARD-OUTPUT 0.5))
 (concept-attribute (MEANINGFUL-MNEMONIC YES 1.0))
 (concept-attribute (NUMBER-OF-FLAGS 4 0.5))
 (concept-attribute (FUNCTION SET-TERMINAL-OPTIONS 0.5))
 (concept-attribute (DOMAIN TERMINAL-CONTROL-INFO 0.5))
 (concept-attribute (RANGE TERMINAL-CONTROL-INFO 0.5))
 (concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.5))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 0 0.5))
 (concept-attribute (PROCESSING-TIME VERY-SHORT 0.5))
 (concept-attribute (OUTPUT-DEVICE PRIMARY-MEMORY 0.5))
 (concept-attribute (NUMBER-OF-FLAGS 11 0.5))
 (instance (CAT 0.3421052631))
 (instance (STTY 0.3157894737))

description of the concept (C-5):

(concept-attribute (FUNCTION FIND-SPELLING-ERRORS 1.0))
 (concept-attribute (DOMAIN FILE 1.0))
 (concept-attribute (RANGE LIST-OF-SPELLING-ERRORS 1.0))
 (concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
 (concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 1.0))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0))
 (concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
 (concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
 (concept-attribute (OUTPUT-DEVICE STANDARD-OUTPUT 1.0))
 (concept-attribute (MEANINGFUL-MNEMONIC YES 1.0))
 (concept-attribute (NUMBER-OF-FLAGS 6 1.0))
 (instance (SPELL 1.0))

description of the concept (C-4):

```
(concept-attribute (FUNCTION TEXT-FORMATING 1.0))
(concept-attribute (DOMAIN FILE 1.0))
(concept-attribute (RANGE FORMATED-FILE 0.75))
(concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.75))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0))
(concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (MEANINGFUL-MNEMONIC NO 0.75))
(concept-attribute (FUNCTION TEXT-FORMATING-AND-TYPESETTING 0.5))
(concept-attribute (OUTPUT-DEVICE GRAPHICS-SYSTEMS-PHOTOTYPESETTER 0.5))
(concept-attribute (NUMBER-OF-FLAGS 14 0.5))
(instance (NROFF 0.6875))
(instance (STYLE 0.5))
(instance (TROFF 0.8125))
(instance (TROFF-T 0.8125))
```

description of the concept (C-2):

```
(concept-attribute (FUNCTION TEXT-EDITING 0.6666666665))
(concept-attribute (DOMAIN USER-FILE 0.6666666665))
(concept-attribute (RANGE USER-FILE 0.6666666665))
(concept-attribute (TYPE-OF-PARAMETERS USER-FILE 0.6666666665))
(concept-attribute (PROCESSING-TIME SHORT 0.6666666665))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (OUTPUT-DEVICE DISK 1.0))
(concept-attribute (FUNCTION EDITING 1.0))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.6666666665))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 0 0.6666666665))
(concept-attribute (MEANINGFUL-MNEMONIC NO 0.6666666665))
(instance (ED 0.575757576))
(instance (EX 0.757575758))
(instance (VI 0.4545454546))
```

description of the concept (C-10):

```
(concept-attribute (FUNCTION OFF-LINE-PRINTING 0.5))
(concept-attribute (DOMAIN FILE 1.0))
(concept-attribute (RANGE PRINTED-FILE 1.0))
(concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.5))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5))
(concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (OUTPUT-DEVICE LINE-PRINTER 1.0))
(concept-attribute (MEANINGFUL-MNEMONIC YES 1.0))
(concept-attribute (NUMBER-OF-FLAGS 20 0.5))
(concept-attribute (FUNCTION PRINTING 1.0))
(concept-attribute (FUNCTION FILE-PRINTING 0.5))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.5))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5))
(concept-attribute (NUMBER-OF-FLAGS 0 0.5))
(instance (LPR 0.625))
```

(instance (PRINT 0.625))

description of the concept (C-11):

```
(concept-attribute (FUNCTION FILE-PRINTING 1.0))
(concept-attribute (DOMAIN FILE 1.0))
(concept-attribute (RANGE PRINTED-FILE 1.0))
(concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 1.0))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 1 1.0))
(concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (OUTPUT-DEVICE LASER-PRINTER 1.0))
(concept-attribute (MEANINGFUL-MNEMONIC NO 1.0))
(concept-attribute (NUMBER-OF-FLAGS 6 1.0))
(concept-attribute (FUNCTION PRINTING 1.0))
(instance (SPIT-I 1.0))
```

description of the concept (C-12):

```
(concept-attribute (FUNCTION FILE-PRINTING 1.0))
(concept-attribute (DOMAIN FILE 1.0))
(concept-attribute (RANGE PRINTED-FILE 1.0))
(concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 1.0))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 1.0))
(concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (OUTPUT-DEVICE PRINTER 1.0))
(concept-attribute (MEANINGFUL-MNEMONIC YES 1.0))
(concept-attribute (NUMBER-OF-FLAGS 9 1.0))
(concept-attribute (FUNCTION PRINTING 1.0))
(instance (PR 1.0))
```

BUILDING A HIERARCHY

LEVEL 4 - number of concepts evaluated at this level: 1

description of the concept (C-9):

```
(NUMBER-OF-FLAGS 0 0.1666666666)
(NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.1666666666)
(NUMBER-OF-FLAGS 20 0.1666666666)
(OUTPUT-DEVICE LINE-PRINTER 0.3333333333)
(FUNCTION OFF-LINE-PRINTING 0.1666666666)
(NUMBER-OF-FLAGS 6 0.3333333333)
(MEANINGFUL-MNEMONIC NO 0.3333333333)
(OUTPUT-DEVICE LASER-PRINTER 0.3333333333)
(NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5)
(FUNCTION PRINTING 1.0)
```

(NUMBER-OF-FLAGS 9 0.3333333333)
 (MEANINGFUL-MNEMONIC YES 0.6666666665)
 (OUTPUT-DEVICE PRINTER 0.3333333333)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FILE-PRINTING 0.8333333335)

removed (attribute, value) pairs with the relevance below the threshold

hierarchical links of the concept (C-9):

(SUBORDINATE-CONCEPT (C-10 0.727272727))
 (SUBORDINATE-CONCEPT (C-11 0.7424242427))
 (SUBORDINATE-CONCEPT (C-12 0.8030303027))

description of the concept (C-9):

(concept-attribute (FUNCTION FILE-PRINTING 0.8333333335))
 (concept-attribute (DOMAIN FILE 1.0))
 (concept-attribute (RANGE PRINTED-FILE 1.0))
 (concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
 (concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5))
 (concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
 (concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
 (concept-attribute (MEANINGFUL-MNEMONIC YES 0.6666666665))
 (concept-attribute (FUNCTION PRINTING 1.0))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5))
 (subordinate-concept ((C-12) 0.8030303027))
 (subordinate-concept ((C-11) 0.7424242427))
 (subordinate-concept ((C-10) 0.727272727))

BUILDING A HIERARCHY

LEVEL 3 - number of concepts evaluated at this level: 1

description of the concept (C-8):

(NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5)
 (FUNCTION PRINTING 1.0)
 (MEANINGFUL-MNEMONIC YES 0.6666666665)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5)

(NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FILE-PRINTING 0.8333333335)

hierarchical links of the concept (C-8):
 (SUBORDINATE-CONCEPT (C-9 0.7575757573))

description of the concept (C-8):
 (concept-attribute (FUNCTION FILE-PRINTING 0.8333333335))
 (concept-attribute (DOMAIN FILE 1.0))
 (concept-attribute (RANGE PRINTED-FILE 1.0))
 (concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
 (concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5))
 (concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
 (concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
 (concept-attribute (MEANINGFUL-MNEMONIC YES 0.6666666665))
 (concept-attribute (FUNCTION PRINTING 1.0))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5))
 (subordinate-concept ((C-9) 0.7575757573))

BUILDING A HIERARCHY *****

LEVEL 2 - number of concepts evaluated at this level: 1

description of the concept (C-7):
 (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5)
 (FUNCTION PRINTING 1.0)
 (MEANINGFUL-MNEMONIC YES 0.6666666665)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FILE-PRINTING 0.8333333335)

hierarchical links of the concept (C-7):
 (SUBORDINATE-CONCEPT (C-8 0.7575757573))

description of the concept (C-7):
 (concept-attribute (FUNCTION FILE-PRINTING 0.8333333335))

```
(concept-attribute (DOMAIN FILE 1.0))
(concept-attribute (RANGE PRINTED-FILE 1.0))
(concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5))
(concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (MEANINGFUL-MNEMONIC YES 0.6666666665))
(concept-attribute (FUNCTION PRINTING 1.0))
(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5))
(subordinate-concept ((C-8) 0.7575757573))
```

BUILDING A HIERARCHY

LEVEL 1 - number of concepts evaluated at this level: 1

description of the concept (C-3):
 (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5)
 (FUNCTION PRINTING 1.0)
 (MEANINGFUL-MNEMONIC YES 0.6666666665)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME INPUT-DEPENDENT 1.0)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335)
 (TYPE-OF-PARAMETERS FILE 1.0)
 (RANGE PRINTED-FILE 1.0)
 (DOMAIN FILE 1.0)
 (FUNCTION FILE-PRINTING 0.8333333335)

hierarchical links of the concept (C-3):
 (SUBORDINATE-CONCEPT (C-7 0.7575757573))

description of the concept (C-3):
 (concept-attribute (FUNCTION FILE-PRINTING 0.8333333335))
 (concept-attribute (DOMAIN FILE 1.0))
 (concept-attribute (RANGE PRINTED-FILE 1.0))
 (concept-attribute (TYPE-OF-PARAMETERS FILE 1.0))
 (concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.8333333335))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.5))
 (concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 1.0))
 (concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
 (concept-attribute (MEANINGFUL-MNEMONIC YES 0.6666666665))
 (concept-attribute (FUNCTION PRINTING 1.0))
 (concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.5))
 (subordinate-concept ((C-7) 0.7575757573))

BUILDING A HIERARCHY

LEVEL 0 - number of concepts evaluated at this level: 1

description of the concept (C-1):

(FUNCTION EDITING 0.2)
 (OUTPUT-DEVICE DISK 0.2)
 (PROCESSING-TIME SHORT 0.1333333333)
 (RANGE USER-FILE 0.1333333333)
 (FUNCTION TEXT-EDITING 0.1333333333)
 (NUMBER-OF-OPTIONAL-PARAMETERS 1 0.1)
 (FUNCTION PRINTING 0.2)
 (RANGE PRINTED-FILE 0.2)
 (FUNCTION FILE-PRINTING 0.1666666667)
 (NUMBER-OF-FLAGS 14 0.1)
 (OUTPUT-DEVICE GRAPHICS-SYSTEMS-PHOTOTYPESETTER 0.1)
 (FUNCTION TEXT-FORMATING-AND-TYPESETTING 0.1)
 (MEANINGFUL-MNEMONIC NO 0.2833333332)
 (RANGE FORMATED-FILE 0.15)
 (FUNCTION TEXT-FORMATING 0.2)
 (NUMBER-OF-FLAGS 6 0.2)
 (PROCESSING-TIME INPUT-DEPENDENT 0.6)
 (TYPE-OF-PARAMETERS FILE 0.6)
 (RANGE LIST-OF-SPELLING-ERRORS 0.2)
 (DOMAIN FILE 0.6)
 (FUNCTION FIND-SPELLING-ERRORS 0.2)
 (NUMBER-OF-FLAGS 11 0.1)
 (OUTPUT-DEVICE PRIMARY-MEMORY 0.1)
 (PROCESSING-TIME VERY-SHORT 0.1)
 (NUMBER-OF-OPTIONAL-PARAMETERS 0 0.2333333332)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.6166666667)
 (RANGE 'TERMINAL-CONTROL-INFO 0.1)
 (DOMAIN TERMINAL-CONTROL-INFO 0.1)
 (FUNCTION SET-TERMINAL-OPTIONS 0.1)
 (NUMBER-OF-FLAGS 4 0.1)
 (MEANINGFUL-MNEMONIC YES 0.5333333333)
 (OUTPUT-DEVICE STANDARD-OUTPUT 0.3)
 (INPUT-DEVICE STANDARD-INPUT 1.0)
 (PROCESSING-TIME MEDIUM 0.1)
 (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.6)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 1 0.2333333332)
 (TYPE-OF-PARAMETERS USER-FILE 0.2333333332)
 (RANGE FILE 0.1)
 (DOMAIN USER-FILE 0.2333333332)
 (FUNCTION CATENATE-AND-PRINT 0.1)

removed (attribute, value) pairs with the relevance below the threshold

hierarchical links of the concept (C-1):

```

(SUBORDINATE-CONCEPT (C-2 0.1428571428))
(SUBORDINATE-CONCEPT (C-3 0.5670634923))
(SUBORDINATE-CONCEPT (C-4 0.5517857145))
(SUBORDINATE-CONCEPT (C-5 0.65))
(SUBORDINATE-CONCEPT (C-6 0.3059523809))

```

description of the concept (C-1):

```

(concept-attribute (NUMBER-OF-OPTIONAL-PARAMETERS 100 0.6))
(concept-attribute (INPUT-DEVICE STANDARD-INPUT 1.0))
(concept-attribute (MEANINGFUL-MNEMONIC YES 0.5333333333))
(concept-attribute (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.6166666667))
(concept-attribute (DOMAIN FILE 0.6))
(concept-attribute (TYPE-OF-PARAMETERS FILE 0.6))
(concept-attribute (PROCESSING-TIME INPUT-DEPENDENT 0.6))
(subordinate-concept ((C-6) 0.3059523809))
(subordinate-concept ((C-5) 0.65))
(subordinate-concept ((C-4) 0.5517857145))
(subordinate-concept ((C-3) 0.5670634923))
(subordinate-concept ((C-2) 0.1428571428))

```

process is FINISHED.

No applicable rules.

=>

APPENDIX G

SAMPLE RUN 2:

GIVEN CONSTRAINTS

=> run

CLUSTERING PROCESS

there are 14.0 events in the initial set

inferred (FUNCTION EDITING 1.0) triplet
in the description of VI

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of TROFF-T

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of TROFF

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of STYLE

inferred (FUNCTION PRINTING 1.0) triplet
in the description of SPIT-I

inferred (FUNCTION PRINTING 1.0) triplet
in the description of PRINT

inferred (FUNCTION PRINTING 1.0) triplet
in the description of PR

inferred (FUNCTION PRINTING 1.0) triplet
in the description of LPR

inferred (FUNCTION EDITING 1.0) triplet
in the description of EX

inferred (FUNCTION EDITING 1.0) triplet
in the description of ED

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION PRINTING): given constraint

remove from the event LPR description the (att, val) pairs
covered by (FUNCTION PRINTING): given constraint

remove from the event PR description the (att, val) pairs
covered by (FUNCTION PRINTING): given constraint

remove from the event PRINT description the (att, val) pairs
covered by (FUNCTION PRINTING): given constraint

remove from the event SPIT-I description the (att, val) pairs
covered by (FUNCTION PRINTING): given constraint

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION EDITING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION PRINTING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION FILE-PRINTING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION TEXT-FORMATING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION FIND-SPELLING-ERRORS):

climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION SIGN-ON):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION SET-TERMINAL-OPTIONS):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (DOMAIN USER-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (RANGE USER-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (RANGE PRINTED-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (INPUT-DEVICE STANDARD-INPUT):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (OUTPUT-DEVICE LINE-PRINTER):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (OUTPUT-DEVICE LASER-PRINTER):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0):
climbing a domain hierarchy

removed (FUNCTION PRINTING 0.9) from the goal-rel-att list:
given constraint

removed (FUNCTION FILE-PRINTING 0.9) from the goal-rel-att list:
already covered

removed (FUNCTION PRINTING 1.0) from the event LPR description:
given constraint

removed (FUNCTION PRINTING 1.0) from the event PR description:
given constraint

removed (FUNCTION PRINTING 1.0) from the event PRINT description:
given constraint

removed (FUNCTION PRINTING 1.0) from the event SPIT-I description:
given constraint

removed (FUNCTION FILE-PRINTING 1.0) from the description of SPIT-I:
already covered

removed (FUNCTION FILE-PRINTING 1.0) from the description of PRINT:
already covered

removed (FUNCTION FILE-PRINTING 1.0) from the description of PR:
already covered

removed (FUNCTION OFF-LINE-PRINTING 1.0) from the description of LPR:
already covered

removed (FUNCTION SET-TERMINAL-OPTIONS 0.05) from the goal-rel-att list:
low relevance

removed (FUNCTION SIGN-ON 0.2) triplet from the goal-rel-att list:
doesn't cover any event

number of generated goal-related (att, val) pairs = 10

LEVEL 0 - ATTEMPT 1

concept: (C-1) - classifying attribute: FUNCTION

following concept is defined by the attribute FUNCTION
and value EDITING:

(C-2) with an instance ED
(C-2) with an instance EX
(C-2) with an instance VI

following concept is defined by the attribute FUNCTION
and value TEXT-FORMATING:

- {C-4} with an instance NROFF
- {C-4} with an instance STYLE
- {C-4} with an instance TROFF
- {C-4} with an instance TROFF-T

following concept is defined by the attribute FUNCTION
and value FIND-SPELLING-ERRORS:

- {C-5} with an instance SPELL

following concept is defined by the attribute FUNCTION
and value OTHER:

- {C-3} with an instance CAT
- {C-3} with an instance LPR
- {C-3} with an instance PR
- {C-3} with an instance PRINT
- {C-3} with an instance SPIT-I
- {C-3} with an instance STTY

there are 4.0 concepts

cardinality of the concept {C-2} is 3.0

cardinality of the concept {C-4} is 4.0

cardinality of the concept {C-5} is 1.0

cardinality of the concept {C-3} is 6.0

heuristic criterion DISTRIBUTION-OF-EVENTS is going to be applied

minimal cardinality = 1.75

UNSUCCESSFUL clustering => initialization process

LEVEL 1 - ATTEMPT 2

concept: {C-3} - classifying attribute: DOMAIN

following concept is defined by the attribute DOMAIN
and value USER-FILE:

- {C-6} with an instance CAT

following concept is defined by the attribute DOMAIN
and value OTHER:

- {C-7} with an instance LPR
- {C-7} with an instance PR

(C-7) with an instance PRINT
 (C-7) with an instance SPIT-I
 (C-7) with an instance STTY

there are 5.0 concepts

cardinality of the concept (C-7) is 5.0
 cardinality of the concept (C-6) is 1.0
 cardinality of the concept (C-2) is 3.0
 cardinality of the concept (C-4) is 4.0
 cardinality of the concept (C-5) is 1.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 2 - ATTEMPT 3

concept: (C-7) - classifying attribute: RANGE

following concept is defined by the attribute RANGE
and value USER-FILE:

following concept is defined by the attribute RANGE
and value PRINTED-FILE:

(C-8) with an instance LPR
 (C-8) with an instance PR
 (C-8) with an instance PRINT
 (C-8) with an instance SPIT-I

following concept is defined by the attribute RANGE
and value OTHER:

(C-9) with an instance STTY

there are 6.0 concepts

cardinality of the concept (C-9) is 1.0
 cardinality of the concept (C-8) is 4.0
 cardinality of the concept (C-6) is 1.0
 cardinality of the concept (C-2) is 3.0
 cardinality of the concept (C-4) is 4.0
 cardinality of the concept (C-5) is 1.0

minimal cardinality = 1.166666667

UNSUCCESSFUL clustering => initialization process

LEVEL 3 - ATTEMPT 4

concept: (C-8) - classifying attribute: INPUT-DEVICE

following concept is defined by the attribute INPUT-DEVICE
and value STANDARD-INPUT:

(C-10) with an instance LPR
(C-10) with an instance PR
(C-10) with an instance PRINT
(C-10) with an instance SPIT-I

following concept is defined by the attribute INPUT-DEVICE
and value OTHER:

there are 6.0 concepts
cardinality of the concept (C-10) is 4.0
cardinality of the concept (C-9) is 1.0
cardinality of the concept (C-6) is 1.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0

minimal cardinality = 1.166666667

UNSUCCESSFUL clustering => initialization process

LEVEL 4 - ATTEMPT 5

concept: (C-10) - classifying attribute: OUTPUT-DEVICE

following concept is defined by the attribute OUTPUT-DEVICE
and value LINE-PRINTER:

(C-11) with an instance LPR
(C-11) with an instance PRINT

following concept is defined by the attribute OUTPUT-DEVICE
and value LASER-PRINTER:
(C-12) with an instance SPIT-I

following concept is defined by the attribute OUTPUT-DEVICE

and value OTHER:
(C-13) with an instance PR

there are 8.0 concepts
 cardinality of the concept (C-13) is 1.0
 cardinality of the concept (C-12) is 1.0
 cardinality of the concept (C-11) is 2.0
 cardinality of the concept (C-9) is 1.0
 cardinality of the concept (C-6) is 1.0
 cardinality of the concept (C-2) is 3.0
 cardinality of the concept (C-4) is 4.0
 cardinality of the concept (C-5) is 1.0

minimal cardinality = 0.875

heuristic criterion DISTRIBUTION-OF-EVENTS is satisfied

clustering process is COMPLETED SUCCESSFULLY

instances of the concept (C-5) are:
SPELL

instances of the concept (C-4) are:
NROFF
STYLE
TROFF
TROFF-T

instances of the concept (C-2) are:
ED
EX
VI

instances of the concept (C-6) are:
CAT

instances of the concept (C-9) are:
STTY

instances of the concept (C-11) are:
LPR
PRINT

instances of the concept (C-12) are:
SPIT-I

instances of the concept (C-13) are:
PR

APPENDIX H

SAMPLE RUN:

GDN UPDATE

=> run

CLUSTERING PROCESS *****

there are 14.0 events in the initial set

inferred (FUNCTION EDITING 1.0) triplet
in the description of VI

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of TROFF-T

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of TROFF

inferred (FUNCTION TEXT-FORMATING 1.0) triplet
in the description of STYLE

inferred (FUNCTION PRINTING 1.0) triplet
in the description of SPIT-I

inferred (FUNCTION PRINTING 1.0) triplet
in the description of PRINT

inferred (FUNCTION PRINTING 1.0) triplet
in the description of PR

inferred (FUNCTION PRINTING 1.0) triplet
in the description of LPR

inferred (FUNCTION EDITING 1.0) triplet
in the description of EX

inferred (FUNCTION EDITING 1.0) triplet
in the description of ED

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION EDITING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION PRINTING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION FILE-PRINTING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION TEXT-FORMATING):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION FIND-SPELLING-ERRORS):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION SIGN-ON):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (FUNCTION SET-TERMINAL-OPTIONS):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (DOMAIN USER-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (RANGE USER-FILE):

climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (RANGE PRINTED-FILE):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (INPUT-DEVICE STANDARD-INPUT):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (OUTPUT-DEVICE LINE-PRINTER):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (OUTPUT-DEVICE LASER-PRINTER):
climbing a domain hierarchy

remove from the goal-rel-att list the (att, val) pairs
covered by (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0):
climbing a domain hierarchy

removed (FUNCTION FILE-PRINTING 0.9) from the goal-rel-att list:
already covered

removed (FUNCTION SET-TERMINAL-OPTIONS 0.05) from the goal-rel-att list:
low relevance

value of the goal-dependent-attribute threshold = 0.1

removed (FUNCTION SIGN-ON 0.2) triplet from the goal-rel-att list:
doesn't cover any event
=> update the GDN

number of generated goal-related (att, val) pairs = 11

LEVEL 0 - ATTEMPT 1

concept: (C-1) - classifying attribute: FUNCTION

following concept is defined by the attribute FUNCTION

and value EDITING:
 (C-2) with an instance ED
 (C-2) with an instance EX
 (C-2) with an instance VI

following concept is defined by the attribute FUNCTION
 and value PRINTING:
 (C-3) with an instance LPR
 (C-3) with an instance PR
 (C-3) with an instance PRINT
 (C-3) with an instance SPIT-I

following concept is defined by the attribute FUNCTION
 and value TEXT-FORMATING:
 (C-4) with an instance NROFF
 (C-4) with an instance STYLE
 (C-4) with an instance TROFF
 (C-4) with an instance TROFF-T

following concept is defined by the attribute FUNCTION
 and value FIND-SPELLING-ERRORS:
 (C-5) with an instance SPELL

following concept is defined by the attribute FUNCTION
 and value OTHER:
 (C-6) with an instance CAT
 (C-6) with an instance STTY

there are 5.0 concepts
 cardinality of the concept (C-2) is 3.0
 cardinality of the concept (C-3) is 4.0
 cardinality of the concept (C-4) is 4.0
 cardinality of the concept (C-5) is 1.0
 cardinality of the concept (C-6) is 2.0

heuristic criterion DISTRIBUTION-OF-EVENTS is going to be applied

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 1 - ATTEMPT 2

concept: (C-3) - classifying attribute: DOMAIN

following concept is defined by the attribute DOMAIN
and value USER-FILE:

following concept is defined by the attribute DOMAIN
and value OTHER:

(C-7) with an instance LPR
(C-7) with an instance PR
(C-7) with an instance PRINT
(C-7) with an instance SPIT-I

there are 5.0 concepts
cardinality of the concept (C-7) is 4.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 2 - ATTEMPT 3

concept: (C-7) - classifying attribute: RANGE

following concept is defined by the attribute RANGE
and value USER-FILE:

following concept is defined by the attribute RANGE
and value PRINTED-FILE:

(C-8) with an instance LPR
(C-8) with an instance PR
(C-8) with an instance PRINT
(C-8) with an instance SPIT-I

following concept is defined by the attribute RANGE
and value OTHER:

there are 5.0 concepts
cardinality of the concept (C-8) is 4.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 3 - ATTEMPT 4

concept: (C-8) - classifying attribute: INPUT-DEVICE

following concept is defined by the attribute INPUT-DEVICE
and value STANDARD-INPUT:

(C-9) with an instance LPR
(C-9) with an instance PR
(C-9) with an instance PRINT
(C-9) with an instance SPIT-I

following concept is defined by the attribute INPUT-DEVICE
and value OTHER:

there are 5.0 concepts
cardinality of the concept (C-9) is 4.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 4 - ATTEMPT 5

concept: (C-9) - classifying attribute: OUTPUT-DEVICE

following concept is defined by the attribute OUTPUT-DEVICE
and value LINE-PRINTER:

(C-10) with an instance LPR
(C-10) with an instance PRINT

following concept is defined by the attribute OUTPUT-DEVICE
and value LASER-PRINTER:
(C-11) with an instance SPIT-I

following concept is defined by the attribute OUTPUT-DEVICE
and value OTHER:
(C-12) with an instance PR

there are 7.0 concepts
cardinality of the concept (C-12) is 1.0
cardinality of the concept (C-11) is 1.0
cardinality of the concept (C-10) is 2.0
cardinality of the concept (C-2) is 3.0
cardinality of the concept (C-4) is 4.0
cardinality of the concept (C-5) is 1.0
cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.0

heuristic criterion DISTRIBUTION-OF-EVENTS is satisfied

clustering process is COMPLETED SUCCESSFULLY

instances of the concept (C-6) are:
CAT
STTY

instances of the concept (C-5) are:
SPELL

instances of the concept (C-4) are:
NROFF
STYLE
TROFF
TROFF-T

instances of the concept (C-2) are:
ED
EX
VI

instances of the concept (C-10) are:
LPR
PRINT

instances of the concept (C-11) are:
SPIT-I

instances of the concept (C-12) are:
PR

=>increase the strength of the rule that posted the triplet
(OUTPUT-DEVICE LASER-PRINTER 0.6)

=>increase the strength of the rule that posted the triplet
(OUTPUT-DEVICE LINE-PRINTER 0.9)

=>increase the strength of the rule that posted the triplet
(FUNCTION FIND-SPELLING-ERRORS 0.7)

=>increase the strength of the rule that posted the triplet
(FUNCTION TEXT-FORMATING 0.8)

=>increase the strength of the rule that posted the triplet
(FUNCTION PRINTING 0.9)

=>increase the strength of the rule that posted the triplet
(FUNCTION EDITING 1.0)

=>decrease the strength of the rule that posted the triplet
(DOMAIN USER-FILE 1.0)

=>decrease the strength of the rule that posted the triplet
(RANGE USER-FILE 1.0)

=>decrease the strength of the rule that posted the triplet
(RANGE PRINTED-FILE 0.9)

=>decrease the strength of the rule that posted the triplet
(INPUT-DEVICE STANDARD-INPUT 0.95)

new goal-rel-att list:

(FUNCTION EDITING 1.0)
(OUTPUT-DEVICE LASER-PRINTER 0.654545455)
(OUTPUT-DEVICE LINE-PRINTER 0.9818181815)
(FUNCTION FIND-SPELLING-ERRORS 0.7636363637)
(FUNCTION TEXT-FORMATING 0.8727272726)
(FUNCTION PRINTING 0.9818181815)
(DOMAIN USER-FILE 0.909090909)
(RANGE USER-FILE 0.909090909)
(RANGE PRINTED-FILE 0.8181818184)

(INPUT-DEVICE STANDARD-INPUT 0.8636363638)
 (NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.4)

CLUSTERING PROCESS

LEVEL 0 - ATTEMPT 1

concept: (C-1) - classifying attribute: FUNCTION

following concept is defined by the attribute FUNCTION
 and value EDITING:

{C-2} with an instance ED
 {C-2} with an instance EX
 {C-2} with an instance VI

following concept is defined by the attribute FUNCTION
 and value PRINTING:

{C-3} with an instance LPR
 {C-3} with an instance PR
 {C-3} with an instance PRINT
 {C-3} with an instance SPIT-I

following concept is defined by the attribute FUNCTION
 and value TEXT-FORMATING:

{C-4} with an instance NROFF
 {C-4} with an instance STYLE
 {C-4} with an instance TROFF
 {C-4} with an instance TROFF-T

following concept is defined by the attribute FUNCTION
 and value FIND-SPELLING-ERRORS:

{C-5} with an instance SPELL

following concept is defined by the attribute FUNCTION
 and value OTHER:

{C-6} with an instance CAT
 {C-6} with an instance STTY

there are 5.0 concepts

cardinality of the concept {C-2} is 3.0

cardinality of the concept {C-3} is 4.0

cardinality of the concept {C-4} is 4.0

cardinality of the concept (C-5) is 1.0
 cardinality of the concept (C-6) is 2.0

heuristic criterion DISTRIBUTION-OF-EVENTS is going to be applied

minimal cardinality = 1.4

UNSUCCESSFUL clustering => initialization process

LEVEL 1 - ATTEMPT 2

concept: (C-3) - classifying attribute: OUTPUT-DEVICE

following concept is defined by the attribute OUTPUT-DEVICE
 and value LINE-PRINTER:
 (C-7) with an instance LPR
 (C-7) with an instance PRINT

following concept is defined by the attribute OUTPUT-DEVICE
 and value LASER-PRINTER:
 (C-8) with an instance SPIT-I

following concept is defined by the attribute OUTPUT-DEVICE
 and value OTHER:
 (C-9) with an instance PR

there are 7.0 concepts
 cardinality of the concept (C-9) is 1.0
 cardinality of the concept (C-8) is 1.0
 cardinality of the concept (C-7) is 2.0
 cardinality of the concept (C-2) is 3.0
 cardinality of the concept (C-4) is 4.0
 cardinality of the concept (C-5) is 1.0
 cardinality of the concept (C-6) is 2.0

minimal cardinality = 1.0

heuristic criterion DISTRIBUTION-OF-EVENTS is satisfied

clustering process is COMPLETED SUCCESSFULLY

instances of the concept (C-6) are:

CAT
STTY

instances of the concept (C-5) are:

SPELL

instances of the concept (C-4) are:

NROFF
STYLE
TROFF
TROFF-T

instances of the concept (C-2) are:

ED
EX
VI

instances of the concept (C-7) are:

LPR
PRINT

instances of the concept (C-8) are:

SPIT-I

instances of the concept (C-9) are:

PR

=>increase the strength of the rule that posted the triplet
(OUTPUT-DEVICE LASER-PRINTER 0.655)

=>increase the strength of the rule that posted the triplet
(OUTPUT-DEVICE LINE-PRINTER 0.982)

=>increase the strength of the rule that posted the triplet
(FUNCTION FIND-SPELLING-ERRORS 0.764)

=>increase the strength of the rule that posted the triplet
(FUNCTION TEXT-FORMATING 0.873)

=>increase the strength of the rule that posted the triplet
(FUNCTION PRINTING 0.982)

=>increase the strength of the rule that posted the triplet
(FUNCTION EDITING 1.0)

new goal-rel-att list:

(FUNCTION EDITING 1.0)
(FUNCTION PRINTING 1.0)
(OUTPUT-DEVICE LINE-PRINTER 1.0)
(OUTPUT-DEVICE LASER-PRINTER 0.7145454544)
(FUNCTION FIND-SPELLING-ERRORS 0.8334545456)
(FUNCTION TEXT-FORMATING 0.9523636363)
(DOMAIN USER-FILE 0.91)
(RANGE USER-FILE 0.91)
(RANGE PRINTED-FILE 0.818)
(INPUT-DEVICE STANDARD-INPUT 0.864)
(NUMBER-OF-NON-OPTIONAL-PARAMETERS 0 0.4)

BIBLIOGRAPHY

Amarel, S., "Program Synthesis as a Theory Formation Task: Problem Representation and Solution Methods," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.

Anderson, J. R., "The Architecture of Cognition," Cambridge, Massachusetts: Harvard University Press, 1983.

Barsalou, L. W., "Determinants of Graded Structure in Categories," Unpublished doctoral dissertation, Stanford University, 1981.

Barsalou, L. W., "Ad Hoc Categories," *Memory & Cognition* **11** (1983) 211-217.

Carbonell, J. G., "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.

Carbonell, J. G., Michalski, R. S., and Mitchell, T. M., "An Overview of Machine Learning," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.

Clancey, W. J., "Heuristic Classification," *Artificial Intelligence* **27** (1985) 289-350.

Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," in *Proceedings of the Workshop on High Level Tools for Knowledge-Based Systems*, Shawnee State Park, OH, October 6-8, 1986.

Collins, A. M. and Loftus, E. F., "A Spreading-Activation Theory of Semantic Memory," *Psychological Review*, **82** (1975) 407-428.

DeJong, G., "An Approach to Learning from Observations," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.

Dietterich, T. G., and Michalski, R. S., "A Comparative Review of Selected Methods for Learning from Examples," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.

Dietterich, T. G., and Michalski, R. S., "Learning to Predict Sequences," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.

Fahlman, S., "NETL: A System for Representing and Using Real-World

Knowledge," MIT Press, Cambridge, Mass., 1979.

Fisher, D., "A Hierarchical Conceptual Clustering Algorithm," *Technical Report*, Department of Information and Computer Science, University of California, Irvine, 1984.

Fisher, D. and Langley, P., "Approaches to Conceptual Clustering," *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp 691-697, Los Angeles, Calif., 1985.

Fried, L. S. and Holyoak, K. J., "Induction of Category Distributions: A Framework for Classification Learning," *Journal of Experimental Psychology: Learning, Memory, and Cognition*, **10** (1984) 234-257.

Haas, N. and Hendrix, G. G., "Learning by Being Told: Acquiring Knowledge for Information Management," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.

Hadžikadić, M., Yun, D. Y. Y., and Ho, W. P. -C., "Characterization of Application Domains for the Expert System Technology," in *Proceedings of the Workshop on High Level Tools for Knowledge-Based Systems*, Shawnee State Park, OH, October 6-8, 1986.

Hadžikadić, M. and Yun, D. Y. Y., "Concept Formation by Psychologically Plausible Classification," to appear in *Proceedings of the Second International Symposium on Methodologies for Intelligent Systems (ISMIS) '87 Colloquia*, Charlotte, North Carolina, October 14-18, 1987.

Holland, J. H., "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., Thagard, P. R., "Induction: Processes of Inference, Learning, and Discovery," The Mit Press, Cambridge, Mass., 1986.

Johnson-Laird, P. N., "Mental Models," Cambridge, Massachusetts: Harvard University Press, 1983.

Kodratoff, Y., and Ganascia, J. G., "Improving the Generalization Step in Learning," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.

Langley, P. and Sage, S., "Conceptual Clustering as Discrimination Learning," *Proceedings of the Fifth Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, 1984.

Langley, P., Zytkow, J. M., Simon, H. A., and Brandshaw, G. L., "The Search for Regularity: Four Aspects of Scientific Discovery," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.

- Langley, P. and Carbonell, J. G., "Language Acquisition and Machine Learning," *Technical Report 86-12*, University of California, Irvine, Calif., 1986.
- Lebowitz, M., "Generalization from Natural Language Text," in *Cognitive Science*, **7** 1 (1983) 1-40.
- Lenat, D. B., "The Role of Heuristics in Learning by Discovery: Three Case Studies," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.
- Michalski, R. S., "Pattern Recognition as Rule-Guided Inductive Inference," in *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **PAMI-2** (1980) 349-361.
- Michalski, R. S., "A Theory and Methodology of Inductive Learning," in
- Michalski, R. S., and Stepp, R. E., "Learning From Observation: Conceptual Clustering," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.
- Michalski, R. S., "Understanding the Nature of Learning: Issues and Research Directions," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.
- Mitchell, T. M., "Version Spaces: An Approach to Concept Learning," Doctoral Dissertation, Stanford University, December, 1978.
- Mitchell, T. and Keller, R., "Goal Directed Learning," in *Proceedings of the Second International Machine Learning Workshop*, Urbana, Illinois, June, 1983.
- Minsky, M., "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, P. Winston (Ed.), McGraw-Hill, New York, 1975.
- Murphy, G. L. and Medin, D. L., "The Role of Theories in Conceptual Coherence," *Psychological Review* **92** (1985) 289-316.
- Palmer, S. E., "Fundamental Aspects of Cognitive Representation," in *Cognition and Categorization*, E. Rosch and B. B. Lloyd (Eds.), Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1978.
- Prieto-Diaz, R. and Freeman, P., "Classifying Software for Reusability," *IEEE Software* January 1987, 6-16.
- Quinlan, J. R., "Learning Efficient Classification Procedures and their Application to Chess End Games," in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.
- Quinlan, J. R., "The Effect of Noise on Concept Learning," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T.

- M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.
- Rosch, E., "On the Internal Structure of Perceptual and Semantic Categories," in *Cognitive Development and the Acquisition of Language*, T. E. Moore (Eds.), Academic Press, New York, 1973.
- Rosch, E. and Mervis, C. B., "Family Resemblances: Studies in the Internal Structure of Categories," *Cognitive Psychology* 7 (1975) 573-605.
- Rosch, E., Mervis, C. B., Gray, W., Johnson, D., and Boyes-Braem, P., "Basic Objects in Natural Categories," *Cognitive Psychology* 7 (1976) 573-605.
- Rosch, E., "Principles of Categorization," in *Cognition and Categorization*, E. Rosch and B. B. Lloyd (Eds.), Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1978.
- Sammut, C., and Banerji, R. B., "Learning Concepts by Asking Questions," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.
- Schank, R. C. and Abelson, R. P., "Scripts, Plans, Goals and Understanding," Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1977.
- Schank, R. C., "Dynamic Memory: A Theory of Reminding and Learning in Computers and People," Cambridge University Press, Cambridge, 1982.
- Schneider, W. and Shiffrin, R. M., "Controlled and Automatic Human Information Processing. I. Detection, Search and Attention," *Psychological Review* 84 (1977) 1-66.
- Shiffrin, R. M. and Schneider, W., "Controlled and Automatic Human Information Processing. II. Perceptual Learning, Automatic Attending, and a General Theory," *Psychological Review* 84 (1977) 127-190.
- Simon, H. A., "Why Should Machines Learn?" in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1983.
- Stepp, R. E. III, "Conjunctive Conceptual Clustering: A Methodology and Experimentation," Doctoral Dissertation, University of Illinois at Urbana-Champaign, 1984.
- Stepp, R. E. III, and Michalski, R. S., "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.
- Tversky, A. and Gati, I., "Studies of Similarity," in *Cognition and Categorization*, E. Rosch and B. B. Lloyd (Eds.), Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1978.
- Utgoff, P. E., "Shift of Bias for Inductive Concept Learning," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and

- T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.
- Vere, S. A., "Induction of Concepts in the Predicate Calculus," *Proceedings of the Fourth International Conference on Artificial Intelligence*, IJCAI, Tbilisi, USSR, 1975.
- Vere, S. A., "Inductive Learning of Relational Productions," in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (Eds.), Academic Press, Inc., New York, 1978.
- Watanabe, S., "Pattern Recognition: Human and Mechanical," John Wiley & Sons, Inc., 1985.
- Waterman, D. A., "Generalized Learning Techniques for Automating the Learning of Heuristics," *Artificial Intelligence* **1** (1970) 121-170.
- Wattenmaker, W. D., Dewey, G. I., Murphy, T. D., Medin, D. L., "Linear Separability and Concept Learning: Context, Relational Properties, and Concept Naturalness," *Cognitive Psychology* **18** (1986) 158-194.
- Winston, P. H., "Learning by Augmenting Rules and Accumulating Censors," in *Machine Learning II: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), Morgan Kaufmann Publishes, Inc., Los Altos, CA, 1986.